Physics of Language Models: Part 4.1, Architecture Design and the Magic of Canon Layers

Zeyuan Allen-Zhu
zeyuanallenzhu@meta.com
FAIR at Meta

May 2, 2025

(version 1.1)*

Abstract

Understanding architectural differences in language models is challenging, especially at academic-scale pretraining (e.g., 1.3B parameters, 100B tokens), where results are often dominated by noise and randomness. To overcome this, we introduce controlled synthetic pretraining tasks that isolate and evaluate core model capabilities. Within this framework, we discover *Canon layers*: lightweight architectural components—named after the musical term "canon"—that promote horizontal information flow across neighboring tokens. Canon layers compute weighted sums of nearby token representations and integrate seamlessly into Transformers, linear attention, state-space models, or any sequence architecture.

We present 12 key results. This includes how Canon layers enhance reasoning depth (e.g., by 2×), reasoning breadth, knowledge manipulation, etc. They lift weak architectures like NoPE to match RoPE, and linear attention to rival state-space models like Mamba2—validated both through synthetic tasks and real-world academic-scale pretraining. This synthetic playground offers an *economical, principled path* to isolate core model capabilities often obscured at academic scales. Equipped with infinite high-quality data, it may even *predict* how future architectures will behave as training pipelines improve—e.g., through better data curation or RL-based post-training—unlocking deeper reasoning and hierarchical inference.

*V1 appeared on SSRN on this date; V1.1 (May 18) improves writing and adds the relu^2 experiments. Project page: physics.allen-zhu.com.

ZA sincerely thanks Vahab Mirrokni for the invitation to the Yale workshop in October 2023, where this research was sparked through enlightening discussions with Vahab Mirrokni and Peilin Zhong. Canon layers build on the idea of uniform attention previously explored in joint work with Yuanzhi Li [3]. ZA thanks Alberto Alfarano for introducing the papers [30, 44, 63, 78], and the PyTorch scaled dot product attention function. At Meta, we extend our heartfelt gratitude to Lin Xiao and Kristin Lauter for their insightful discussions and unwavering supports, which made this research possible. Special thanks go to Wangzhi Dai, Dinesh Kannappan, Niki Kim, Junjie Qian, Ammar Rizvi, Travis Seevers, and Stephen Hartken at Meta, as well as Abraham Leal from W&B; without their invaluable technical assistance, the experiments presented in this paper would not have been feasible. We are deeply grateful to Songlin Yang and Ali Behrouz for providing detailed instructions on replicating their academic-scale pretraining experiments, and Fangcheng Sun for many helpful conversations on architecture design in general.

Contribution statement. ZA proposed all ideas, conducted all investigations, implemented all code, performed all experiments, authored the entire manuscript, and managed all necessary compliance reviews and social promotions; the term *Canon Layers* was jointly conceived and designed with Xiaoli Xu.

1 Introduction

Recent advances in large language models (LLMs) have sparked transformative progress across numerous tasks, including question answering, summarization, translation, code generation [13, 15, 39, 61]. Despite rapid progress, systematic understanding of effective neural architecture design has remained elusive, fundamentally hindered by some major challenges.

Challenge 1: Pretraining loss as an unreliable proxy for intelligence. Architectural comparisons often rely on perplexity or cross-entropy loss, but these metrics do not reliably reflect real-world capabilities—especially since natural data is *skills-mixed*. For example, state-space architectures like Mamba [19, 26] frequently achieve lower perplexity early in training due to rapid memorization, yet perform poorly on complex reasoning tasks. Reliance on *early stopping via perplexity* is thus problematic: it may lead to comparing models that have merely internalized surface-level linguistic patterns without developing deeper reasoning or factual understanding [31].

Challenge 2: Noise below emergence thresholds. Emergent abilities—complex skills that only arise in large-scale models (e.g., 7B parameters, 10T tokens [1])—complicate architectural comparisons at smaller, academic scales (e.g., 1.3B parameters, 100B tokens [9, 25, 69]). At these scales, small benchmark gains (e.g., 2%) often result from random initialization or data shuffling—variance that can cause 2–4% swings in accuracy (see Figure 1). More fundamentally, **models fail even the simplest 2-hop reasoning** tasks, performing no better than random guessing.[1] This basic reasoning floor *masks architectural differences* in more advanced cognitive skills, making evaluation at this scale deeply unreliable. While large-scale industry training might reveal these differences, its prohibitive cost blocks systematic ablations, impeding academic contributions to rigorous architecture science—and often reducing design choices to heuristics and guesswork.

Challenge 3: Grokking, Data Quality, and Curriculum Learning. Failures in complex reasoning tasks typically stem from deficiencies in training data, *not* architectural limitations. Too few challenging samples and a lack of intermediate-complexity data often force models to rely on unstable grokking behavior—where generalization only emerges after unnecessarily long pretraining [43]—and disrupt curriculum learning [10]. For instance, models lacking 2-hop reasoning data may unpredictably learn 3-hop tasks after extensive exposure to 1-hop and 3-hop examples. This makes training highly sensitive to randomness, further complicating architectural comparisons. Reinforcement learning (RL)-based post-training methods, such as GRPO [53] and PPO [52], aim to address this by delivering tailored data at optimal difficulty levels. While effective, these methods introduce new experimental confounds—it becomes unclear whether performance gains stem from pretraining, RL fine-tuning, stochastic training dynamics, or architectural strength.

Our approach: Atomic decomposition of intelligence. To overcome the noise and cost of real-world pretraining—especially at academic scales where even 2-hop reasoning fails to emerge—we decompose intelligence into atomic components, such as reasoning depth and breadth, and design synthetic, controllable *pretrain* tasks to isolate and evaluate them independently. This framework sharply characterizes architectural strengths and scalability under clean, idealized conditions (see Figure 1), offering a principled and economical path for architecture design.

This approach directly addresses Challenge 1 by enabling *single-skill evaluations*, minimizing the confounding factors prevalent in real-world pretraining data. For example, it allows rigorous comparisons of whether architecture A outperforms architecture B in reasoning depth, while ensuring modifications do not degrade other capabilities. By isolating intrinsic architectural biases,

[1] In our simplest 2-hop reasoning tasks, birth years for 3 individuals are presented, followed by 3 "[name2] was born in the same year as [name1]" equivalences. The model is prompted to infer the second group's birth years. Academic-scale models can only guess. See Result 12.

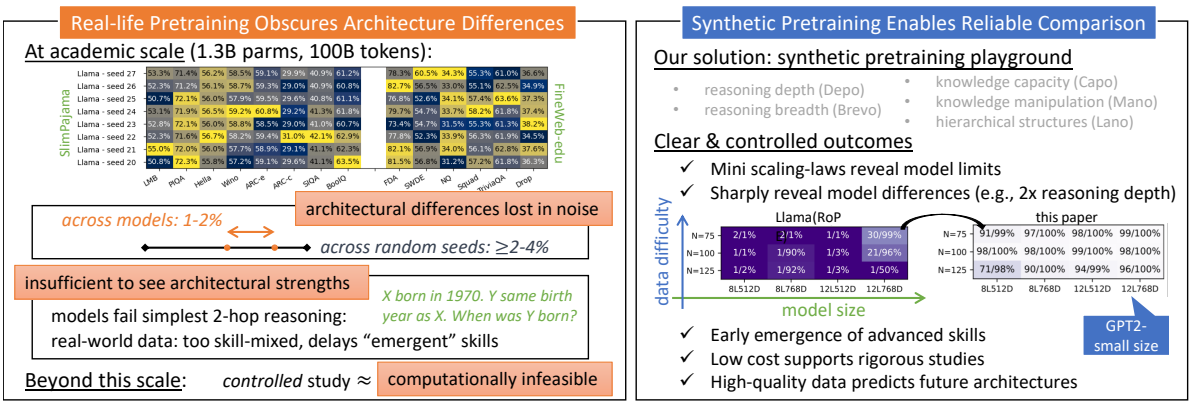


Figure 1: Architecture search in noisy real-life pretraining (good luck!) vs. our synthetic playground (scientific rigor).

synthetic *pretrain* tasks reveal properties often obscured by noise and mixed signals in typical real-life setups.

Challenge 2 is mitigated by *lowering resource* needs for rigorous comparisons. Synthetic benchmarks yield infinite high-quality data, enabling meaningful pretraining even for smaller models (e.g., GPT2-small) where complex skills might otherwise not emerge. In these controlled environments, capabilities like deep multi-hop reasoning *emerge clearly and reliably*, allowing rapid identification of architectural limitations, investigation of *mini scaling-laws*, and uncover trends that real-world pretrained models often fail to reveal due to noise or insufficient signal despite extensive training.

For Challenge 3, we manage data difficulty distributions to ensure adequate representation of intermediate-complexity samples, smoothing learning curves and enabling the *early and consistent emergence* of advanced skills—unlike less predictable real-world data prone to grokking-driven instability. As training pipelines improve—via better data curation or RL-based continued pretraining—synthetic pretrain benchmarks may provide *predictive insight* into which architectures best support scaling to more advanced tasks in the future.

We draw inspiration from physics, where idealized settings—such as frictionless planes or vacuum chambers—reveal first principles by removing confounding factors. Similarly, synthetic tasks eliminate the noise, randomness, and data contamination of real-world datasets, enabling clean, controlled, apples-to-apples architectural comparisons, much like Galileo's Pisa tower experiment.

This paper's key contributions are summarized below:

Result 0: Building the Synthetic Playground (Section 2+3). We introduce five synthetic pretraining tasks—Depo (reasoning depth), Brevo (reasoning breadth), Capo (knowledge capacity), Mano (knowledge manipulation), and Lano (hierarchical language structure). This controlled environment can reveal clear, commonsense capability trends *at smaller scales*: linear attention (e.g., GLA [68]) consistently underperforms; state-space models like Mamba2 [19] excel at memory but struggle with reasoning; and full Transformers dominate on complex reasoning tasks.

Result 1: Canon Layers Add Horizontal Information Flow (Section 4). Transformers lack horizontal information flow within layers, leading to inefficiencies even on simple tasks like associative recall. Drawing on the musical canon (overlapping repetition), we introduce *Canon layers*, horizontal "residual links" across neighboring tokens that can be flexibly inserted at multiple points — before attention (Canon-A), inside attention (Canon-B), before MLP (Canon-C), inside MLP (Canon-D). While Canon layers can be implemented in many ways—even simple random averaging is highly effective—this paper focuses on trainable 1-d linear convolutions of kernel size 4. This is lightweight and integrates seamlessly into any sequence model with minimal code.

Results 2–5: When Transformer Meets Canon (Section 5).

- Boost performance. In our playground, Canon layers improve reasoning depth (200–400%),

reasoning breadth (30%), knowledge manipulation length (30%), and more. These stem from enhanced hierarchical learning dynamics and come with minimal computational overhead.

- Reviving NoPE. Integrating Canon layers transforms NoPE models into strong performers, often matching or surpassing RoPE(+Canon). Canon layers outperform positional fixes like ALiBi [44] or H-Alibi [30], and reducing/removing RoPE usage improves length generalization.
- Ablation study. Canon layers contribute cumulatively across sublayer positions (Canon-A/B/C/D), independently of attention or MLP components. Residual links improve training efficiency; minimal parameter tuning is required without compromising stability.
- MLP and MoE. Canon layers can recover some knowledge capacity lost in gated MLP or mixture-of-expert (MoE) architectures, via improved training efficiency and stability.

Results 6–7: When Linear Attention Meets Canon (Section 6).

- Boost performance. Canon layers elevate Gated Linear Attention (GLA [68]) from 1-hop to 4-hop reasoning depth, double its reasoning breadth and knowledge manipulation length, making it comparable to Mamba2 and even surpassing it on tasks like Brevo.
- Ablation study. Residual links and full Canon (A/B/C/D) are essential for maximizing effectiveness for linear-attention models, partial implementations may underperform.

Results 8–9: When Mamba Meets Canon (Section 7).

- Secret of success. Mamba2's performance is driven by its built-in `conv1d` mechanism, which acts as a non-linear Canon-B layer applied to selective coordinates. Removing `conv1d` drops performance to match GLA, while replacing it with full Canon layers further boosts results, highlighting the importance of horizontal information flow over SSM design.
- Ablation study. Canon choices—such as integration points and residual links—can influence Mamba2's performance. Mimetic initialization [63], while optimized for length generalization, harms shorter-context tasks, underscoring the need for diverse pretraining environments.

Results 10–11: Comparing Architectures (Section 8).

- Controlled comparisons. Applying full Canon layers consistently across RoPE, NoPE, Mamba2, and GLA allows controlled comparisons, revealing that full transformers outperform linear models in hierarchical reasoning tasks, achieving twice the reasoning depth.
- Reasoning depth challenges. In GLA and Mamba2, limited reasoning depth stems from accumulated compression and retrieval errors—not memory capacity—pinpointing a key focus for future research on linear models. Until this is resolved, hybrid designs (e.g., sliding-window Transformers with linear backbones) remain the most scalable path to deeper reasoning.

Result 12: Academic-Scale Real-World Pretraining (Section 9). Training 1.3B-parameter models on 100B tokens (context length 4096) reveals high noise and limited resolution, making many architectural comparisons statistically unreliable. Still, several consistent patterns emerge. Canon layers significantly improve NoPE and GLA—elevating them to match RoPE and Mamba2, respectively—while removing `conv1d` weakens Mamba2 to GLA level. Linear models lag behind full Transformers on retrieval-heavy tasks, even with Canon layers. All models fail 2-hop reasoning, even in short contexts (e.g., 100 tokens), underscoring the limitations of academic-scale pretraining. Reducing or removing RoPE improves long-context generalization when Canon layers are present. These results align with our synthetic findings (Results 3, 6, 8, 10, 11).

In summary, Canon layers fundamentally improve horizontal information flow across diverse architectures, enabling deeper reasoning and efficient scalability. Combined with synthetic benchmarks, they provide systematic insights into future opportunities in model design.

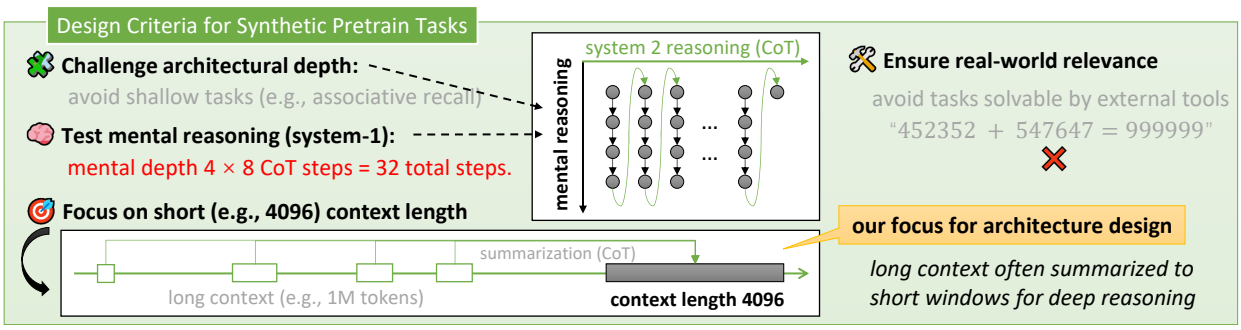


Figure 2: Our design criteria for synthetic pretrain tasks.

2 Synthetic Tasks for Decomposing Intelligence

We design synthetic tasks to systematically evaluate specific capabilities of language model architectures under controlled conditions, minimizing confounds and enabling clean comparisons. Task selection is guided by four criteria:

Criterion 1: Tasks must not be shallow. Shallow tasks—like associative recall or copying—are easily solvable by small and shallow models, and do not meaningfully test architectural strength. Deep learning relies on stacked layers to progressively learn abstract features [4], so tasks involving hierarchical reasoning better evaluate architectural scalability and efficiency.

Criterion 2: Emphasis on mental thinking. Tasks should assess a model's ability to reason internally without Chain-of-Thought (CoT). While CoT helps decompose problems, it does not reflect intrinsic "system 1" reasoning [73]. For example, a model reasoning 4 steps internally and 8 via CoT achieves 32 steps, but *only internal ones reflect architectural strength.* Current models like o3/R1 produce verbose reasoning traces even for trivial prompts (e.g., "Hello")—revealing inefficiencies in system 1. To guide architectural progress, tasks must target mental reasoning.

Criterion 3: Avoid emphasis on length generalization. Length generalization is often unstable—sensitive to random seeds and training order [78]—and thus unreliable for comparing architectures. While length generalization is important, models over-optimized for long contexts (e.g., 100k tokens) may exhibit reduced performance on standard lengths like 4096 tokens.[2] In practice, long inputs are typically summarized into shorter windows before reasoning, so we prioritize evaluating architectures on dense, 4096-token contexts, where critical reasoning unfolds.

Criterion 4: Relevance to real-world skills. Tasks should prioritize broadly applicable skills while avoiding capabilities better suited to external tools. For example, large-number arithmetic (e.g., adding 10-digit numbers) is theoretically interesting but can be delegated to Python interpreters; failures in this area typically reflect limited data exposure rather than architectural weaknesses (e.g., Llama3 70B miscalculates 452352 + 547647). Synthetic tasks should focus on universally relevant skills, aligned with real-world applications, to ensure meaningful assessments.

2.1 Our First Set of Five Synthetic Pretrain Tasks

To operationalize the criteria above, we design five synthetic tasks—each targeting a distinct dimension of language model capability. We name them DEPO, BREO, CAPO, MANO, and LANO.

Task Depo: Mental reasoning depth. Reasoning depth represents a fundamental capability for LLMs, requiring models to retrieve information through multi-step computation. Task DEPO evaluates reasoning depth as k-hop traversal over directed permutations, where models compute the

[2] This is observed in methods like ALiBi [44], Halibi [30], and Mimetic initialization [63], whose performance degrades on shorter contexts, as we show in this paper.

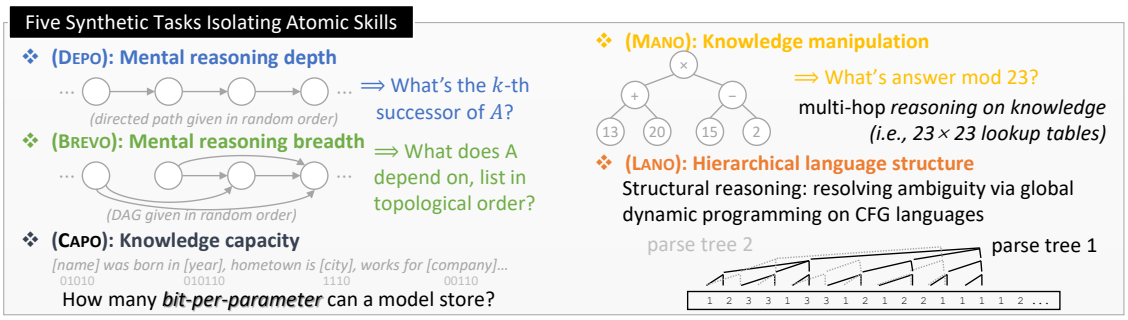


Figure 3: Overview of our five synthetic tasks, each isolating an atomic skill for rigorous architectural comparison.

k-th successor for each query q entirely internally, without intermediate steps like Chain-of-Thought (CoT).[3] Each instance is formatted as:

```
<bos> x1 y1 x2 y2 ... xn yn <query_k1> q1 a1 <query_k2> q2 a2 ... <eos>
```

Here, $2n$ tokens encode n directed edges $x_i \to y_i$, forming a random permutation of n nodes.

The dataset is controlled by two parameters: N, the maximum permutation size, and K, the maximum reasoning depth. During training, n is sampled from $[3, N]$, while $k \in [1, K]$. Context lengths are fixed to 2048 tokens. We employ two variants of DEPO:

- DEPO1: Each node spans 1–2 tokens from vocab size 50, with $N = 225$, 300, 375 and $K = 8$.
- DEPO2: Each node spans 5–7 tokens from vocab size 4, with $N = 75$, 100, 125 and $K = 16$.

Evaluation focuses on both the hardest cases ($n = N, k = K$) and intermediate difficulty ($k = K/2$). For weaker models, we utilize *reduced* training setups with $K = 4$, denoted DEPO1($K = 4$) and DEPO2($K = 4$). The full methodological details are provided in Appendix A.1.

Task Brevo: Mental reasoning breadth. This evaluates a model's ability to process multiple dependencies simultaneously, as required in tasks involving tree-like traversal or dependency graphs. For example, solving queries like "Who are Alice's nephews?" or GSM-like examples requires parallel reasoning across branches of a graph to process relationships bottom-up [71]. Task BREVO isolates this capability using recursive traversal of directed acyclic graphs (DAGs), abstracting away natural language or arithmetic complexities. Each task instance is formatted as:

```
<bos> x1 y1 x2 y2 ... xm ym <query> q <ans> a1 a2 ... ap <eos>
```

Here, $2m$ tokens define m edges $x_i \to y_i$, representing dependencies where y_i depends on x_i. Upon receiving a query vertex q, the model outputs all vertices recursively reachable from q, sorted in topological order starting from the leaves (e.g., $u \to v \to q$ yields output u followed by v).

The dataset is parameterized by N, the maximum graph size, with DAGs created using $n \leq N$ nodes, each of degree at most 4. Pretraining data is sampled by varying graph sizes, while testing focuses on the hardest graphs ($n = N$). We employ two variants of BREVO:

- BREVO1: Each vertex name spans a single token, with $N = 70/90/110$, fit within 1024 tokens.
- BREVO2: Name spans 2–4 tokens of vocab size 4, with $N = 30/40/50$, fit within 1536 tokens.

A key discovery from [71] revealed that, due to the non-uniqueness of valid outputs, language models must preprocess the entire topological order of the DAG *mentally* before generating the first token a_1. This insight confirms that our synthetic data rigorously evaluates reasoning breadth by requiring models to globally process the underlying graph structure before producing outputs.

Task Capo: Knowledge capacity. Task CAPO evaluates a model's efficiency in encoding factual knowledge directly within its parameters, quantified as *bits per parameter*, which measures

[3] Using CoT would reduce the k-hop task to simpler 1-hop associative recall.

reliable storage capacity. Following the framework in [7], synthetic datasets of (fake) biographies are constructed to test knowledge retention. Each biography includes several attributes (e.g., birthdate, university, employer, etc.) and is presented in diverse paraphrased formats to reduce surface-level memorization [5, 6]. Capacity is measured using the next-token prediction distribution, accounting for both exact correctness and partial accuracy.

To highlight architectural differences, we adopt an undertrained regime where each biography is exposed only 100 times during pretraining.[4] The dataset includes $N = 50$K to 2M biographies, encoding 2×10^6 to 10^8 total bits of information. Models of varying sizes are tested, and results are visualized via "bit vs. model size" plots. Additional details are provided in Appendix A.3.

Task Mano: Knowledge manipulation. Task MANO evaluates a distinct form of reasoning: the ability to manipulate stored knowledge internally, contrasting with in-context reasoning tasks like DEPO or BREVO. While those tasks focus on reasoning over external tokens, MANO requires models to retrieve factual knowledge embedded in their parameters and perform hierarchical computation entirely mentally. This combination of retrieval and reasoning makes knowledge manipulation uniquely challenging and a skill that must be learned during pretraining.[5]

To test this capability, MANO employs synthetic modular arithmetic expressions inspired by human mental computation, particularly small-number arithmetic like the 9×9 multiplication table. Models solve multi-step arithmetic problems without intermediate steps like Chain-of-Thought. For example, given: `<bos> + * a b - c d <ans>` the task requires evaluating $((a \times b) + (c - d)) \bmod 23$ for $\ell = 3$, where operands a, b, c, d are sampled uniformly from $[0, 22]$. Modular arithmetic provides the foundational factual knowledge (23×23 operation tables), while the task challenges hierarchical reasoning by recursively composing operations. Additional details are provided in Appendix A.4.

The dataset is parameterized by a maximum expression length L, with ℓ sampled uniformly from $[1, L]$. We prepare three MANO datasets across difficulty levels: $L = 10$, 13, and 16.

Task Lano: Hierarchical language structure. Task LANO evaluates structural reasoning over hierarchical relationships and long-range dependencies. Unlike DEPO, BREVO, and MANO, which rely on explicit key-value pairs (in-context or knowledge), LANO challenges models to infer implicit recursive structures across sequences and resolve global ambiguities within them.

To test this, LANO leverages synthetic datasets built from context-free grammars (CFGs). Training sequences consist of CFG-valid sentences separated by `<bos>` tokens. For example:

`<bos> 3 3 2 2 1 ... 3 3 1 2 <bos> 1 2 3 3 1 ... 1 2 2 1 <bos> ...`

CFGs are designed with token-level ambiguity, where local tokens (e.g., 1, 2, 3) provide insufficient information to directly infer their mapping to CFG rules. Resolving this requires dynamic programming to globally map the entire sequence to a valid recursive application of CFG rules, which must also be learned during training. This reasoning grows in worst-case complexity ($O(n^3)$) as sequence lengths increase. Details are in Appendix A.5.

Building upon cfg3f [3], which includes sequences of lengths 100–500, we introduce extended datasets cfg3j and cfg3k, with sequences ranging up to 200–1000 tokens to increase recursive depth and test models on more nested rules and longer dependencies. Training uses context lengths of 1536 for cfg3j and cfg3k, compared to 512 for cfg3f. Evaluation prompts models with `<bos>` to

[4] Exposing each biography 1000 times during pretraining diminishes architectural differences, as even transformers without MLP layers can achieve similar storage efficiency [7]. Uniform exposure ensures clean systematic comparisons while avoiding confounding effects tied to rare outliers and junk data [7].

[5] For instance, questions like "Was [name] born in an even or odd month?" or derived 2-hop queries such as "What is [name]'s sister's birthdate?" demand reasoning layers over stored knowledge. These skills cannot reliably emerge through supervised fine-tuning alone [6] and require development during pretraining or continued pretraining.

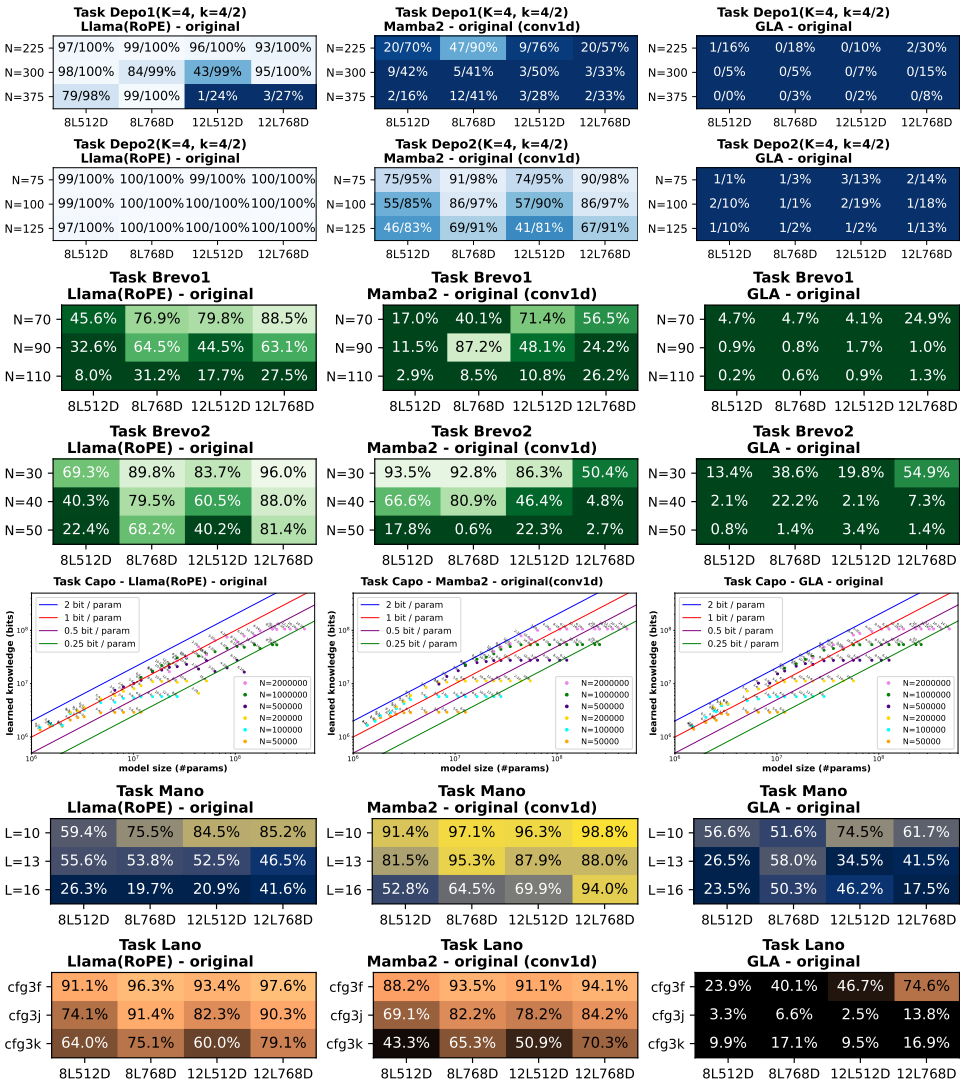


Figure 4: **Initial comparison** of RoPE, Mamba2, and GLA on five synthetic tasks. GLA performs poorly everywhere except knowledge capacity (CAPO); Mamba2 excels at knowledge (CAPO, MANO); Llama(RoPE) is best at reasoning (DEPO, BREVO, LANO). This confirms our synthetic playground **as effective** for architectural comparisons, but introducing Canon layers (see rest of the paper) will build a Pisa tower for more controlled and fair comparisons, where **the landscape shifts drastically** and reasoning depth improves 2–4×.

generate CFG-valid sentences, validated via a dynamic programming parser. KL divergence is also used to compare token distributions against ground truth.

In summary, this set of five synthetic tasks covers non-overlapping skills and distinct aspects of accuracy—token-level (DEPO, MANO), generative (BREVO, LANO), and distributional (CAPO, LANO). While this pool can be further enriched, it serves as a strong starting point for deriving meaningful architectural insights, as demonstrated in the following sections.

3 Initial Comparison on Well-Known Architectures

Language model architectures have evolved significantly since Transformers [64], resulting in three major families distinguished by computational mechanisms.

Quadratic-time attention models, pioneered by the original Transformer, include prominent

architectures such as BERT [35] and GPT2 [46]. Recent refinements include Rotary Position Embeddings (RoPE) [12, 59] and gated MLP layers [54]. We use the Huggingface implementation of Llama, denoted as Llama(RoPE), incorporating RoPE and gated MLP, and a variant without positional embeddings, Llama(NoPE). We refer to these as RoPE and NoPE respectively when clear from the context. We exclude relative positional embeddings due to limited empirical benefits but additional computational costs [3].

RoPE models often generalize poorly beyond training context lengths. In contrast, NoPE generalizes better but suffers from lower overall performance. Recent attention-score modifications (e.g., ALiBi [44] and Hard-Alibi [30]) partially address this trade-off; we discuss in later sections.

Linear-time attention reduces computation by compressing sequences into fixed-length representations. Examples include Linformer [65], Performer [14], Linear Transformer [34]. We focus on more recent Gated Linear Attention (GLA) [68], known for computational efficiency and scalability.

Recurrent and state-space models process long sequences using evolving hidden states instead of attending over all tokens. Mamba [19, 26] exemplifies this category; we analyze its second generation (Mamba2). Other prominent models include S4 [56], S5 [56], RetNet [60], RWKV [42], HGRN [45], GSA [76], DeltaNet [70], and GatedDeltaNet [69].

Avoidance of hybrid architectures. We exclude models integrating attention with linear or state-space methods—e.g., Griffin [20], Samba [48], GatedDeltaNet-H1/H2 [69] or sliding-window attention—to maintain clarity. Such hybrid approaches excel in extremely long contexts (e.g., 1 million tokens), but our analysis focuses explicitly on precision within standard context windows (4096 tokens). In practice, long contexts are often compressed to shorter segments (e.g., via CoTs) for final detailed processing, making precise local reasoning essential.

Hybrid models can *obscure architectural trade-offs*; aggregated results may not reflect individual component contributions clearly. For instance, Mamba2 is strong in memory tasks yet weaker in structured reasoning. Hybrids blending linear/state-space modules with attention can mask these distinctions. Thus, for transparency, this study focuses entirely on isolated architectures to clearly analyze their inherent strengths and weaknesses.

Architecture Size Standardization. To ensure fair comparisons, we standardize model sizes and evaluate Llama, GLA, and Mamba2 as representative architectures from each family.

For all tasks except Capo, we experiment with four architecture sizes. Llama models have 12 or 8 layers, with hidden dimensions of 768 or 512 (and 12 or 8 heads), denoted as `12L768D`, `8L512D`, etc. (`12L768D` matches GPT2-small). We translate these configurations into GLA, Mamba2, and Mamba2(mlp) to ensure comparable parameter counts.[6]

For Capo (bit-per-parameter knowledge capacity), we vary model and data sizes more widely. Following [7], we denote model scale by ℓ-h: for Llama, this means ℓ layers, hidden size $64h$, and h heads. We extend this notation consistently to GLA and Mamba2.

Training. We use identical training settings (batch size, training steps, and learning rate choices) across architectures to ensure fair comparisons. Complete details are provided in Appendix A. We also fix random seeds so that all architectures pre-train on precisely identical data sequences.

3.1 Initial Comparison Results

From Figure 4, linear-attention GLA performs weakest overall, Mamba2 excels in knowledge tasks (Capo, Mano), and Llama(RoPE) performs best on reasoning tasks (Depo, Brevo, Lano).

[6]The original Mamba2 has no MLP layers: each Mamba layer has $6d^2$ parameters (for hidden size d), compared with $12d^2$ in Llama. Thus, we configure Mamba2 with 24 or 16 layers to match Llama's size. Mamba2(mlp) alternates Mamba and gated MLP blocks, thus keeping 12 or 8 total layers. See details in Appendix C.

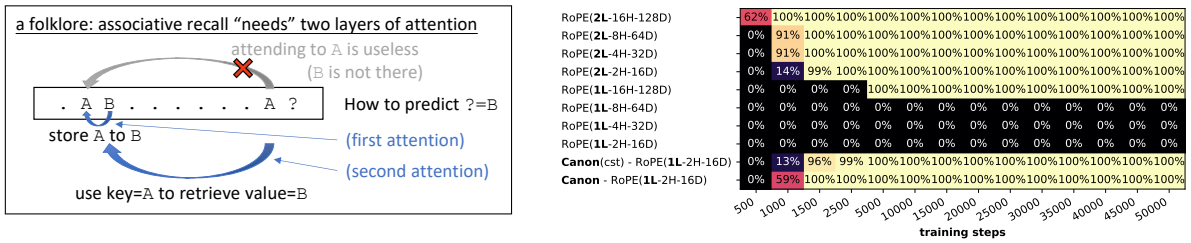


Figure 5: A trivial token-copying experiment for 500 tokens, added for completeness. 1-layer RoPE requires $d \geq 128$, while 2-layer RoPE or 1-layer RoPE + Canon achieves 100% with $d = 16$.

These results validate the effectiveness of our synthetic playground; however, we avoid deeper interpretation at this point. As shown later, Llama and GLA lack a critical architectural component, making this initial comparison incomplete, unfair, and less informative.

For now, we highlight several *key remarks*.

3×4 mini scaling laws. Randomness may affect outcomes. For example, in Task MANO, despite two seeds and four learning rates per configuration, smaller models sometimes outperform larger ones. Thus, robust statistical comparisons are crucial. We address this by testing our synthetic tasks systematically at *three* data scales and *four* architecture sizes (even more for Task CAPO). These "3×4" mini scaling laws enable clearer visual comparisons, reducing variability.

Benefits of synthetic tasks. Synthetic tasks clarify architectural differences starkly (e.g., 90% vs 5%), clearly exposing strengths and weaknesses. By contrast, real-world experiments often produce modest differences (e.g., 2%) buried in noise. Thus, synthetic pretraining environments allow clean evaluations of architectures' scalability and true capabilities.

Interpreting task failures. If a specific architecture (of a given size) fails at a certain difficulty level (e.g., large N or k), it does not imply the model cannot learn the skill given infinite training. Our comparison uses a fixed, limited training budget: all architectures train for the same number of steps with identical data and shuffling, reporting best accuracy across multiple learning rates. Thus, results should be seen as differences in the *speed of skill acquisition*, not absolute capability.[7]

Predicting future pipelines. Synthetic tasks simulate idealized, high-quality pretraining conditions targeting core skills like multi-hop reasoning (DEPO). Unlike datasets such as FineWeb-edu or SlimPajama, which contain sparse reasoning examples obscured by simpler content, synthetic tasks highlight core capabilities. Currently, 100B-token pretraining fails even simplest 2-hop reasoning (Result 12). As training pipelines evolve—via improved data curation or RL-based post-training—synthetic tasks like DEPO may better predict models' potential and guide architectural choices.

4 Canon Layers: Enhancing Horizontal Information Flow

Attention-based Transformers are widely recognized for their ability to perform associative recall—e.g., predicting `?` in the sequence `[A] [B] ... [A] [?]` where `?` = `[B]`. One might expect the second `[A]` could simply attend to the first to retrieve `[B]`, but causal masking makes this impossible: the first occurrence of `[A]` sees no future tokens. Accurate recall thus "requires" two attention layers—the first copies the first `[A]` into its neighbor `[B]`; the second uses this enriched representation, querying by `[A]` to retrieve value = `[B]` (via key = `[A]`). Using global attention just to pass information between adjacent tokens is, in effect, *shooting a bird with a cannon*.

[7] Faster learning is practically important—for example, a model ideally learns reasoning skills quicker than pure memorization. Similar observations arise in knowledge capacity tasks [7], where architectural differences vanish with ample training but become pronounced when training budgets are limited.

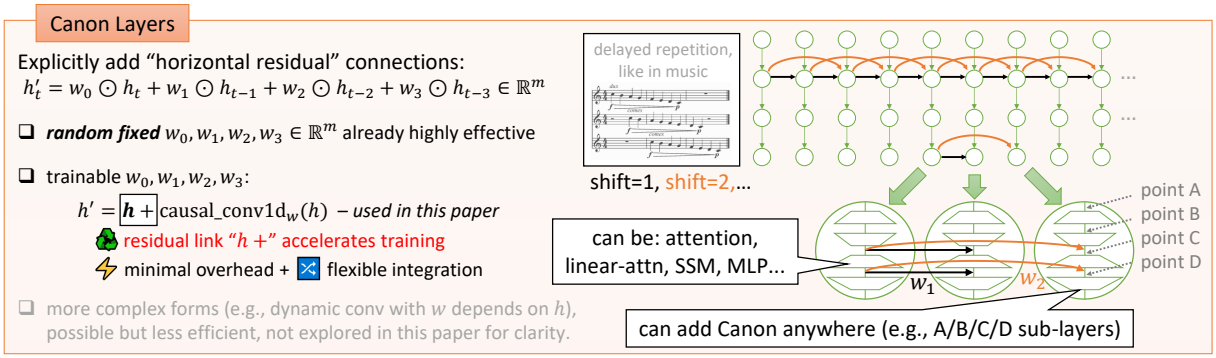


Figure 6: Illustration of Canon layers.

Remark 4.1. This is not a strict lower bound. A 1-layer Transformer is Turing-complete and can perform recall by blindly aggregating most (or all) context into one position, allowing MLP to do local query/key/value computations. But this is inefficient: Figure 5 shows that 1-layer Transformer needs hidden size 128 to recall length-500 sequences, while 2 layers succeed with size 16.

The importance of local context. Even simple tasks like token recall require careful mixing of local context—*not to say* more complex ones or when words span multiple tokens. Since MLP layers don't mix tokens, attention must handle all communication. Rotary and relative positional encodings help by biasing attention toward nearby tokens, but they remain tied to attention and still "shoot birds with cannons." Similar issues arise in GLA [68] and Mamba2, where recent-token information must be retrieved via compression mechanisms not optimized for local detail.

Canon layers: general form. Inspired by (vertical) residual connections, we introduce *Canon layers* to enhance horizontal information flow across neighboring tokens. Canon layers aggregate nearby hidden states into the current position, enabling lightweight local mixing within a fixed window (e.g., size 4), unlike attention-based global aggregation or recurrent compression.

Formally, for any hidden states $h_t \in \mathbb{R}^m$ at token position t, a Canon layer computes:

$$h'_t = w_0 \odot h_t + w_1 \odot h_{t-1} + w_2 \odot h_{t-2} + w_3 \odot h_{t-3},$$

where $\odot$ denotes element-wise multiplication, $w_i \in \mathbb{R}^m$ $(i = 0, 1, 2, 3)$ are weights, and padding zeros are used for boundary conditions. We call this *Canon*, borrowing from the musical term, as it resembles melodies played sequentially at fixed temporal delays.[8]

Flexible Integration. Canon layers integrate at multiple points within each Transformer block:

- *Canon-A*: Before the attention block ($m = d$ if hidden size is d).
- *Canon-B*: Inside the attention block, applied to Q/K/V projections ($m = 3d$).
- *Canon-C*: Before the MLP block ($m = d$).
- *Canon-D*: Within the MLP block ($m = 4d$ for standard MLP, $m = \frac{16}{3}d$ for gated MLP).

Combining all four points gives *Canon-ABCD* (full-score Canon); partial combinations (Canon-A/B/ABC) can also be explored. Canon layers integrate flexibly across diverse architectures, including linear-attention and state-space models. For Mamba2 (without standard MLP layers), Canon layers appear at Canon-A and Canon-B positions (yielding Canon-AB); for Mamba2(mlp), the complete Canon-ABCD applies. Canon-B in Mamba2 scales as $m = 4d + o(d)$.[9]

[8] In Pachelbel's Canon in D, violins sequentially play the same melody with delays, creating overlapping horizontal repetition patterns analogous to Canon layers.

[9] For example, Mamba2 settings with `ssm_state_size=64`, `num_heads=16` result in $m = 4d + 144$ dimensions.

Canon layers: Implementation variants. Canon layers can be implemented in many ways. Even a simple version with fixed, random weights—aggregating past three tokens as *horizontal residual links*—already notably enhances performance (Figure 20, Appendix).[10] More complex variants—e.g., dynamic convolutions with input-dependent weighting—are possible but not studied here, as it remains unclear whether such additional cost is justified.

In this paper, for simplicity and efficiency, we implement Canon layers as 1-d causal convolution with kernel size 4, available through efficient CUDA kernels implemented by the open-source H3 library (pip package `causal_conv1d`) [23]. We also incorporate explicit residual connections:

$$h'_t = h_t + \text{conv1d}\big([h_t, h_{t-1}, h_{t-2}, h_{t-3}]\big) \ , \tag{4.1}$$

denoted as Canon(res). Without residual connections, we denote it Canon(no-res). Minimal code changes (just a few lines) are needed for integration. Even fully enabled (Canon-ABCD), Canon layers increase the parameter count minimally.[11] Our emphasis is on clearly demonstrating Canon layers' substantial performance benefits; detailed runtime optimizations remain future work.

Related Work. A precursor to Canon layers appears in [3], which studied uniform attention—i.e., averaging the past k tokens—for $k \in \{1, 2, 4, 8, 16\}$ on CFG tasks. Surprisingly, this simple mixing outperformed GPT2 with absolute positional embeddings and closely approached GPT2(RoPE).[12] Canon layers generalize this idea: we apply learned, position-specific mixing over a short window (typically 4 tokens), removing value and projection matrices for better efficiency and modularity.

Our use of `causal_conv1d` is inspired by Mamba [19, 26] and GLA [68], which trace back to H3 [23], where the component was introduced as "shift-SSM." After the initial release of our paper, we also became aware of Primer [57], which proposes "multi-dconv-head" attention. These models apply `conv1d` (often with SiLU activation) within SSM or attention modules, without residual connections. In our terminology, these roughly correspond to Canon-B(no-res).

Our work generalizes and isolates this design as the Canon layer, and systematically evaluates its effect across all types of sequential models and sublayers (A/B/C/D). By studying Canon under *controlled* synthetic pretraining, we can clearly attribute performance gains to the conv1d-based mixing mechanism, rather than to other architectural components such as attention or state-space recurrence. Moreover, we show that Canon layers are intrinsically *not tied to attention or SSMs*—and in fact, may not benefit from being tightly coupled to them.

Convolutions have been used in Transformers for different goals. Conformer [27] and CvT [67] integrate heavier convolutional modules for feature extraction in speech and vision. In contrast, Canon layers are lightweight and designed to enhance horizontal information flow—like horizontal "residual links." Notably, even random-weight Canon layers yield substantial improvements.

Concurrent work on Multi-Token Attention (MTA) [25] explores more complex 2D convolutional layers within attention heads. While MTA improves associative recall, it is heavier and more attention-specific. Investigating whether such designs offer further gains when combined with Canon, or whether Canon alone suffices for most settings, is an interesting direction for future work.

[10] Unlike vertical residual links ($h' = h + \sigma(\mathbf{W}h)$), Canon layers aggregate multiple token vectors from different relative positions ($t-1$, $t-2$, $t-3$). Assigning fixed orthogonal directions effectively provides each position a unique "ID" for aggregation. Simple scalar weighting (e.g., $h'_t = h_t + 0.4h_{t-1} + 0.2h_{t-2} + 0.1h_{t-3}$) can degrade performance.

[11] Fewer than 0.45% parameters for GPT2-small. For a 1.3B-parameter Llama with Canon-ABCD enabled, parameters increase by 0.0063%, runtime overhead on an H100 GPU with naive implementation (PyTorch bf16, flash attention, causal conv1d kernels) is 12.4%, 14.1%, and 20.8% for forward, backward, and generation respectively. For Canon-AC, overheads reduce to 5.8%, 5.8%, and 7.0%. Further runtime efficiencies are possible (e.g., consolidating multiple Canon operations across layers), though these optimizations remain beyond this paper's scope.

[12] One ICML reviewer rejected the paper, commenting that the results were "too surprising to be true." We invite curious readers to try it themselves—it really works.

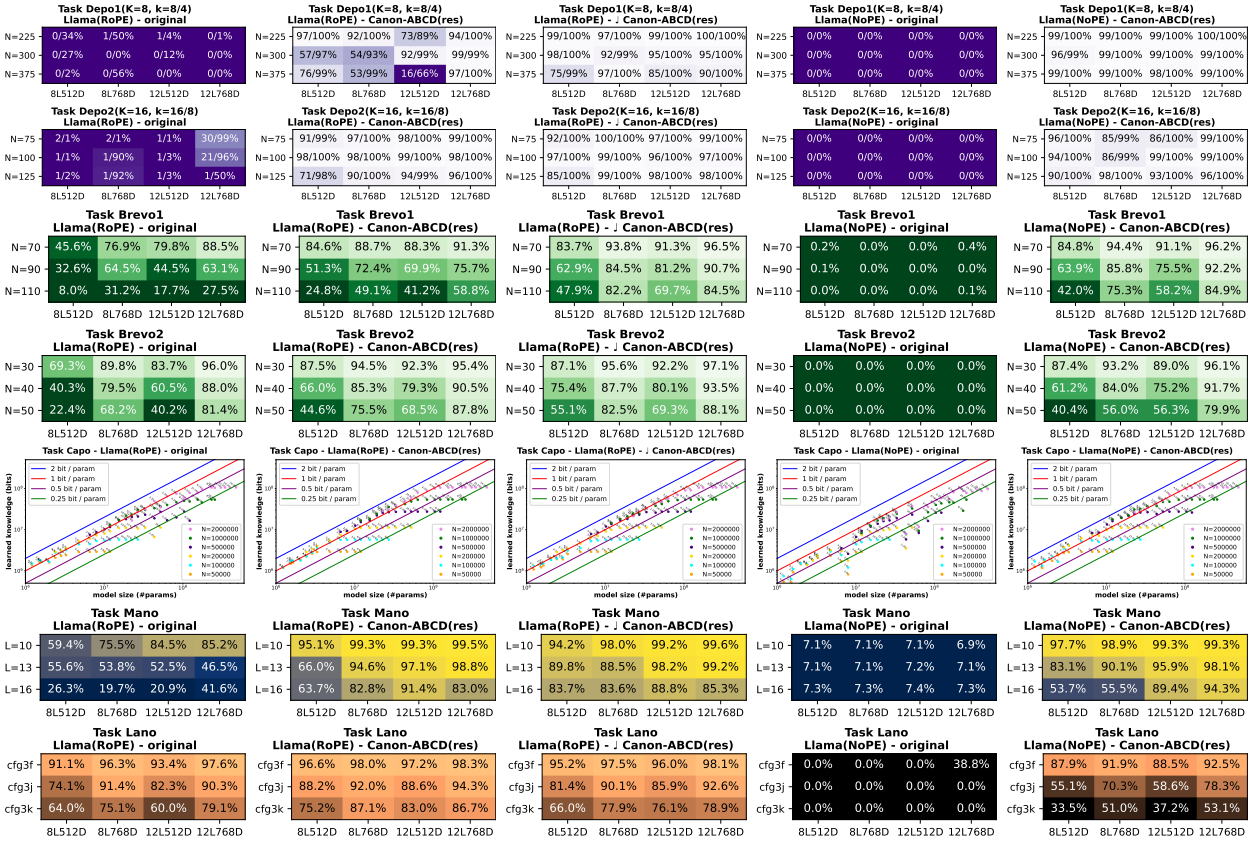


Figure 7: **Column 1→2**: Canon layers dramatically enhance RoPE, improving reasoning depth by 2–4×.
Column 4→5: Canon transforms NoPE into a strong performer on par with RoPE-based models.
Column 2+5→3: With Canon, RoPE usage can be reduced — RoPE + ♩Canon (RoPE enabled for 1/4 dimensions) outperforms both RoPE/NoPE + Canon, great news for length generalization!

Remark. This figure uses DEPO1(K=8) and DEPO2(K=16). Earlier results in Figure 4 were based on DEPO1(K=4) and DEPO2(K=4), because model performances were weaker.

5 When Transformer Meets Canon

Figure 4+7 show that a 12-layer, 768-dimension Llama(RoPE) model trained on our ideal data can only handle 4-hop retrieval in contexts of length 2048. Can this be any better?

5.1 RoPE with Canon Layers

Result 2 (Figure 7 — 1st vs. 2nd column)**.** *In our controlled playground, Canon layers (ABCD) introduce substantial improvements: with a 0.5% increase in trainable parameters, reasoning depth of RoPE increases by 2-4×, reasoning breadth by 30%, knowledge capacity by 10–15%, knowledge manipulation length by 30%, measurable gains in hierarchical language structure reasoning.*

Task Depo. In reasoning depth, RoPE pretrained on DEPO1($K = 8$)—covering ($k \leq 8$)–hop instances—achieves near-zero accuracy even at $k = 4$, whereas RoPE+Canon-ABCD exceeds 50% at $k = 8$. On DEPO2($K = 16$)—a more challenging setup where each directed edge spans 10–14 tokens, far beyond a 4-token Canon window—RoPE completely fails, while RoPE+Canon-ABCD attains near-perfect accuracy at $k = 16$. This demonstrates that Canon layers are not merely for single-token recall: by enriching local representations of multi-token segments, they empower the

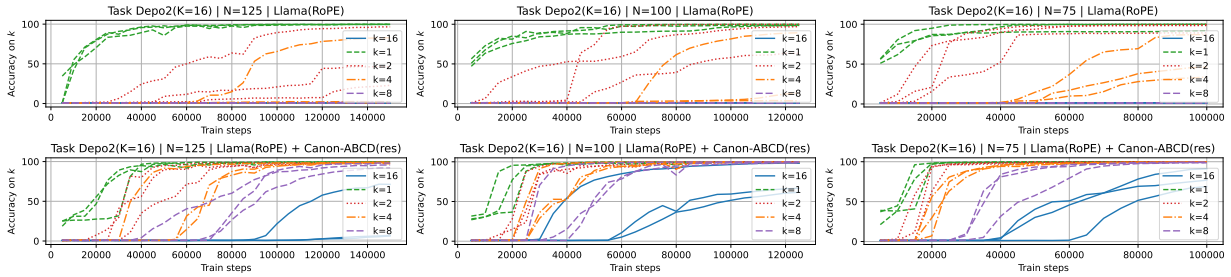


Figure 8: Training curves for the 8L512D RoPE model, with and without Canon layers, on Task DEPO2($K = 16$), evaluated at depths $k = 1, 2, 4, 8, 16$ and graph size $n = N$, across three learning rates.

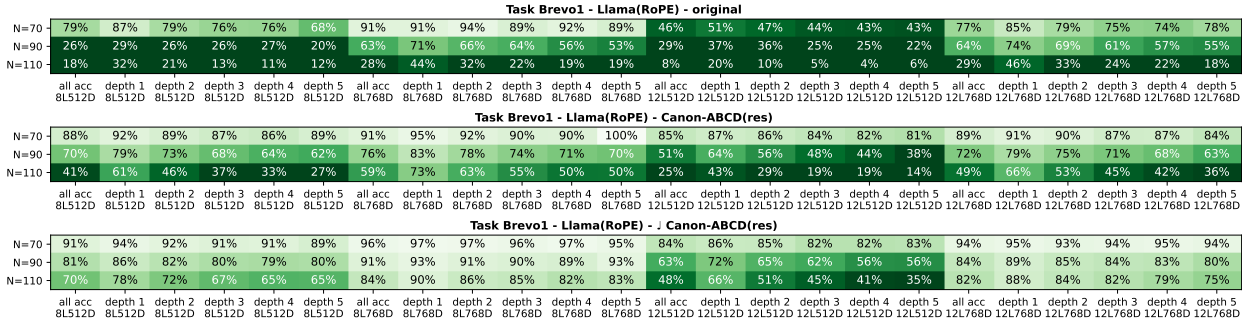


Figure 9: Detailed accuracies for Task BREVO1, shown overall and stratified by dependency graph depths $1, 2, 3, 4, 5$.

global attention to more effectively chain information across hops.[13]

These gains may seem surprising. For associative recall (analogous to DEPO1 with $k = 1$), theory suggests a single Canon + attention layer suffices (recall Figure 5), suggesting Canon could reduce required attention layers by at most one. So, why a 2–4× increase in reasoning depth?

The answer lies in learning dynamics. Deep reasoning tasks like DEPO unfold through a *hierarchical learning* process—models first master 1-hop, then gradually progress to 2-hop, 3-hop, and beyond. This process relies heavily on two factors: (1) training data spanning a range of difficulty levels and (2) architectural support like residual connections. Without either—e.g., training only with $k = 8$ data *or* removing residuals—the model can fails entirely.[14]

Thus, architectures that enable faster mastery of 1- and 2-hop reasoning climb the hierarchy faster, as illustrated in Figure 8. RoPE + Canon-ABCD achieves deeper reasoning progression much faster than vanilla RoPE, leveraging the inherent easy-to-hard structure of multi-hop tasks. We emphasize again that this is not about performance under infinite training data—RoPE could eventually achieve similar accuracy on DEPO2($K = 16$). However, RoPE + Canon achieves comparable results with significantly fewer training steps, making it far more efficient.

Task Brev. On reasoning breadth, we observe 30% improvement by introducing Canon-ABCD. Specifically, the accuracies of RoPE to solve BREVO1($N = 70$) or BREVO2($N = 30$) resemble the performance of RoPE+Canon to solve BREVO1($N = 90$) or BREVO2($N = 40$). Since input length scales with N, this reflects roughly 30% increase in reasoning breath.

To understand the source of this improvement, we analyze the accuracy across tasks stratified by *depth* of the dependency depth. Recall each query in BREVO requires the model to identify all vertices it recursively depends on, forming a sub-DAG of varying (minimum) depth. In Figure 9,

[13] DEPO2 is designed so a 4-token window cannot resolve key–value pairs spanning 10–14 tokens, posing a substantial challenge even for Canon.

[14] The first theory foundation for why deep learning can perform deep (hierarchical) learning was established by Allen-Zhu and Li [2] (in the 3-layer case) and Allen-Zhu and Li [4] (for $\omega(1)$-layer). They show that deep learning relies on easy-to-hard curricula and residual structures for progressively building complexity.

we plot model accuracy not only overall but also separately for problem instaces spanning DAG depths of $1, 2, 3, 4, 5$. The results show that vanilla RoPE struggles with instances involving greater DAG depth, whereas RoPE+Canon improves reasoning performance on deeper structures. This suggests that Canon-ABCD enhances localized reasoning paths within Transformer blocks, allowing for better handling of recursive dependency, which can be challenging for standard attention alone.

Task Capo. On knowledge capacity, prior work [7] found that gated MLP layers in Llama(RoPE) reduce model capacity due to slower and less stable training dynamics. One remedy proposed in that work is to revert gated MLP back to standard MLP; however, this sacrifices reasoning capability (see Section 5.4). Here, we present an alternative solution: adding Canon layers. Canon layers improve training speed and increase the effective capacity by 10–15% in the controlled 100-exposure pretraining regime for CAPO. On a separate note, GPT2(RoPE) models that originally employ standard MLP exhibit no capacity loss after Canon layers are introduced (Figure 11).

Task Mano. On knowledge manipulation, Canon layers increase manipulable length. RoPE+Canon matches the performance of vanilla RoPE on Mano($L = 10$) when tested on Mano($L = 13$), a 30% improvement in length. This again stems from Canon layers accelerating hierarchical learning, enabling the model to scale from simpler tasks ($L = 1$) to more complex ones ($L = 2$, $L = 3$, and beyond) faster. For simplicity, we omit the hierarchical learning speed visualization.

Task Lano. Canon layers improve RoPE's performance on hierarchical language structure reasoning, though interpreting the gains requires some algorithmic background. For context, dataset cfg3k adds one level of structural complexity above cfg3f, using the same CFG rule distribution (see Appendix A.5). RoPE+Canon outperforms standard RoPE on cfg3k, but still struggles to fully handle this increased complexity. This is expected, as deeper CFG structures increase sequence length n by 2–3×, and parsing these CFGs with dynamic programming involves worst-case time complexity $O(n^3)$. Consequently, cfg3k poses arguably more than 8× greater computational challenge compared to cfg3f. Our intermediate dataset cfg3j has difficulty around 4×, suggesting RoPE+Canon can handle roughly twice as challenging structure-learning tasks comparing to RoPE.

Summary. Canon layers consistently improve performance across reasoning, knowledge and language tasks, all without introducing instability or accuracy trade-offs.

5.2 NoPE with Canon Layers

Result 3 (Figure 7+10[a]). *Canon layers transform NoPE. Key findings include:*

- *NoPE+Canon matches RoPE+Canon and even surpasses it on* DEPO*; a remarkable result given that without Canon, NoPE achieves essentially zero performance on all measures.*
- *NoPE+Canon significantly outperforms existing fixes for NoPE, such as ALiBi and H-Alibi.*
- *With Canon layers, RoPE usage can be greatly reduced: RoPE on only 1/4 dims (denoted RoPE+♩Canon) outperforms both RoPE/NoPE+Canon, great news for length generalization.*

[a](Sub-results correspond to Figure 7 (4th vs 5th column), Figure 10, and Figure 7 (3rd column), respectively.)

Canon layers skyrocket NoPE performance. Canon layers dramatically improve NoPE (No Positional Embedding) transformers, lifting them from near-zero accuracy to competitive levels, even slightly surpassing RoPE+Canon on reasoning depth. NoPE-Canon is only weaker on Task LANO, which involves hierarchical structural learning over long sequences, thus relying more heavily on relative distance between input tokens; yet even there NoPE-Canon remains competitive with alternatives such as Mamba2.

Dominance over existing fixes on NoPE. While NoPE excels at length generalization, its performance on complex reasoning tasks has historically been weak. Fixes like ALiBi [44] and Hard-Alibi [30] partially address this: ALiBi applies a distance-based penalty to attention weights[15], while Hard-Alibi disables attention beyond distance h for the h-th head. Although these methods improve NoPE performance (partly mimicking RoPE), Canon layers clearly dominate. As shown in Figure 10 (top), NoPE+Canon significantly outperforms both alternatives.

Minimal RoPE usage with Canon layers. Canon layers eliminate the need for heavy RoPE usage, and excessive RoPE can even hurt performance. With Canons, minimal RoPE usage is sufficient—often preferable—for optimal results. For example, enabling RoPE on half of the heads at half of their dimensions (denoted ♩Canon) consistently outperforms full RoPE usage or NoPE, as shown in Figure 7 (3rd column). **This is great news for long-context generalization**: RoPE is a known bottleneck for Transformers with longer inputs. As Canon layers allow significantly reduced RoPE without performance loss, they become indispensable for length generalization tasks.[16]

Remark 5.1. Despite their versatility, Canon layers alone cannot fully resolve extremely challenging tasks that require deep hierarchical reasoning over long sequences (e.g., cfg3k in Task LANO). Such tasks, requiring $O(n^3)$ dynamic programming over 1000 tokens, remain computationally demanding. Nevertheless, Canon layers consistently offer huge improvements outside these specialized scenarios.

These findings translate to real-life. To be shown in Section 9, NoPE+Canon consistently matches or surpasses RoPE+Canon in real-world pretraining; the RoPE+♩Canon variants outperform RoPE+Canon on several reasoning tasks, particularly involving long-context inputs.

Remark 5.2. This paper focuses on architectural differences within computational stages after relevant information is retrieved into manageable contexts (e.g., 4096 tokens). Techniques like DeepSeek's NSA architecture [74], designed for retrieval and compression from extremely long inputs (e.g., 1M tokens), are complementary to Canon layers. Such techniques and Canon layers can thus jointly handle distinct processing phases in long-context models.

5.3 Ablation Studies With Canon Layers

This section systematically investigates the design choices in Canon layers via ablation studies.

Component-level contributions. Each Canon component (A/B/C/D) contributes meaningfully to performance, with cumulative benefits from combinations. Adding even a single Canon layer yields notable gains, and stacking multiple Canon layers across sub-layers further amplifies these improvements, especially on weaker architectures like NoPE. Summaries appear in Figure 10 (for model size 12L768D) and additional size experiments in Appendix D.2.

Role of residual connections. Residual links around Canon layers — i.e., the "h_t+" part of (4.1) — are critical for training stability and effective learning, preserving vertical computational pathways and allowing global representations to selectively incorporate local context. Without residual connections, training becomes slower and less stable (see rows marked "NoRes" in Figure 10).

Independence of Attention/MLP. Prior works (e.g., the GLA [68] codebase and Primer [57]) focused solely on convolution operations within attention projections — Canon-B(no-res). However, we find that Canon-ACD alone already achieves substantial performance improvements, without modifying attention mechanisms. Similarly, Canon-ABC or even Canon-AC perform strongly

[15] Specifically, adding $-|j-i| \cdot 2^{-8h/H}$ to the logits of head h of H total heads.

[16] Alternative reduced-RoPE configurations explored in the appendix include: ♩♩ (1/4 of heads at full dimensions) and ♩♩♩ (all heads at 1/4 dimensions, as in GPT-NeoX [12]). Among these, ♩ and ♩♩♩ are comparable, slightly better than ♩♩ according to Figure 22.

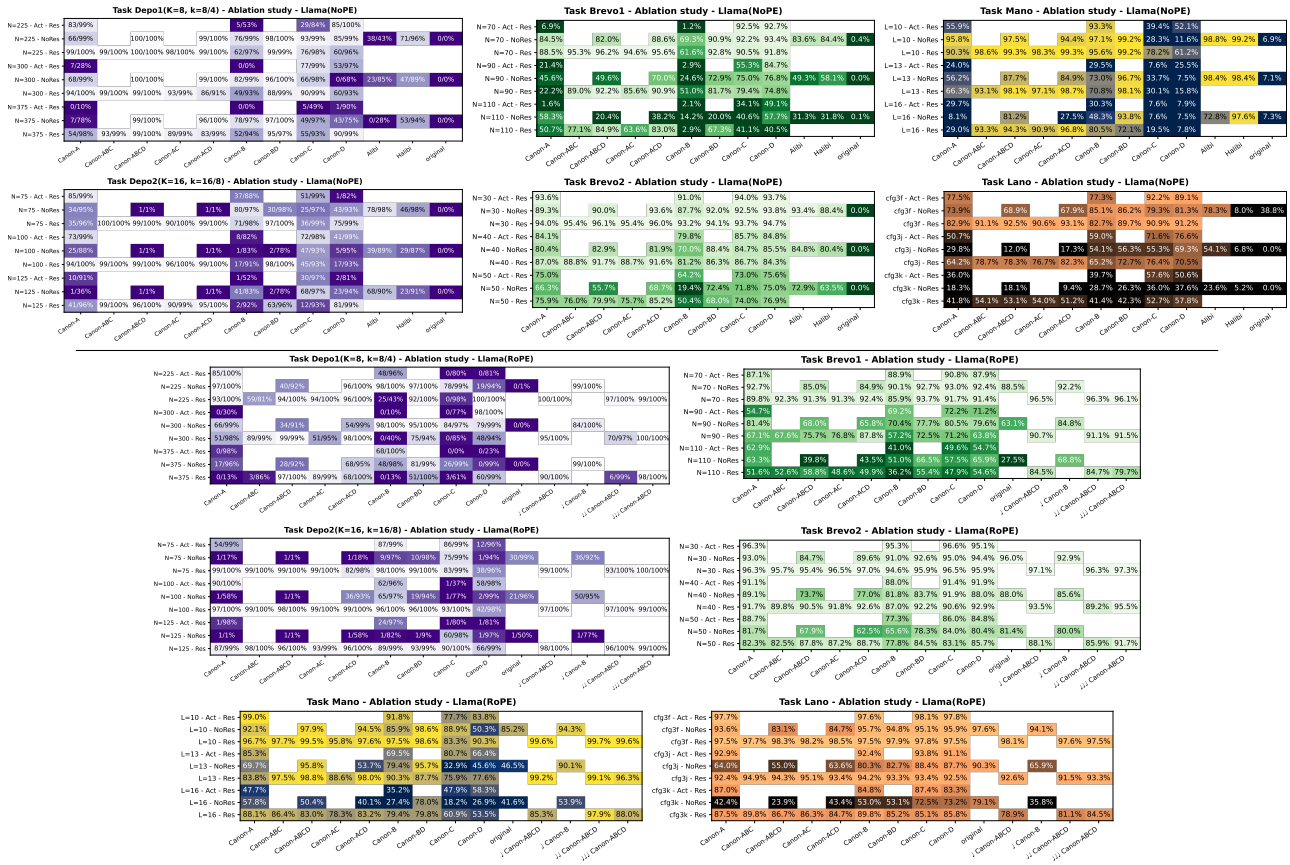


Figure 10: Ablation study on 12-layer, 768-dim Transformers—NoPE (top) and RoPE (bottom)—with Canon variants (A–D), residual links, activation functions, ALiBi, and H-Alibi. Blank entries indicate untested configs due to resource limits. Additional ablation studies (with more model sizes) are in Figure 26 (RoPE), Figure 28 (NoPE), and Figure 27 (RoPE+Primer) in Appendix D.2.

without adjusting MLP layers. They all strongly outperform Canon-B(no-res) and thus outperform Primer. This highlights Canon layers' general role in enhancing horizontal information flow across architecture sub-layers, *independently complementing* attention or MLP mechanisms.

Nonlinear activations and computational simplicity. Contrary to prior works (e.g., H3/Mamba), adding activation functions such as SiLU after the Canon layers does not yield noticeable benefits. Canon layers effectively inject local context directly into token positions, and nonlinear operations are sufficiently handled by the attention and MLP blocks (see rows marked "Act" in Figure 10).

Result 4 (Figure 10)**.** *Canon layers are lightweight, versatile, and effective enhancements that integrate seamlessly into Transformers. Key findings:*

- *Canon-A/B/C/D yield meaningful, cumulative improvements when stacked, and can be flexibly applied anywhere independent of attention or MLP modifications.*
- *Residual connections in Canon design are essential for stable, efficient training.*
- *Adding nonlinear activations (e.g., SiLU) provide no measurable benefit, simplifying design.*

(This **differs from prior works**: we show where to insert Canon layers, how to stabilize them, and why they matter.)

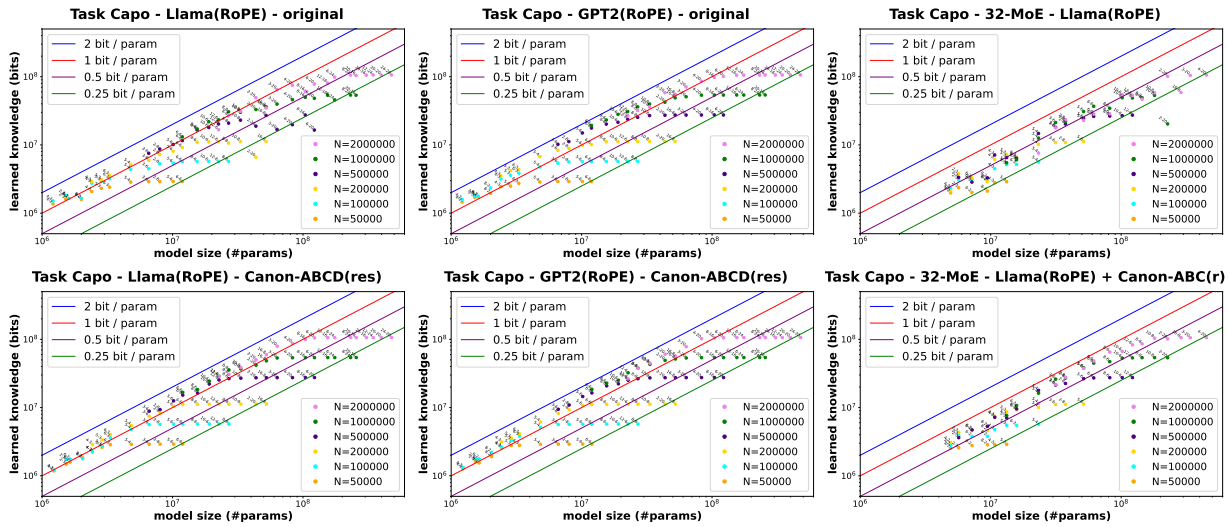


Figure 11: Evaluation of knowledge capacity (Capo) across architectures, measured as bits per parameter. The first row represents baseline models, while the second row shows improvements with Canon layers added. **Conclusion:** Canon layers enhance knowledge storage for architectures that are slower to train, such as gated MLP and MoE, mitigating the capacity gap between gated and standard MLP as identified in [7].

5.4 MLP and Mixture-of-Experts

Our synthetic playground provides a valuable framework to evaluate *broader architectural choices*.

Gated vs. standard MLPs. Gated MLPs [54], which replace standard MLP operations $V\sigma(Wx)$ by $V(\sigma(W_1x)\cdot(W_2x))$, improve expressiveness and parameter efficiency. Widely adopted by large-scale models (e.g., PaLM [15], Llama [61, 62], Mistral [32]), gated MLPs have become standard design choices. However, [7] found that gated MLP reduces knowledge capacity by about 30% in limited-exposure scenarios (e.g., 100-exposure Task Capo) due to slower convergence.

Thus, what is the best tradeoff? Our experiments (Figure 20) confirm gated MLP has slight advantage over standard MLP ("GPT2-style") on reasoning-heavy tasks, showing noticeable improvements on knowledge manipulation (Mano) and smaller gains on reasoning breadth (Brevo). Thus, replacing gated MLP with standard MLP may not be the best choice. However, keep in mind that adding Canon layers already partially mitigates gated MLP's capacity loss (recall Result 2), due to improving training dynamics and speed, recovering about half of its lost capacity.

Mixture-of-Experts. Mixture-of-Experts (MoE) [22, 55] enhances parameter efficiency by replacing dense MLPs with multiple parallel "experts," selectively routing tokens to fewer active experts. While MoE achieves good scalability (particularly on knowledge capacity) and competitive inference-time performance, it suffers from significantly slower knowledge acquisition speed during training. For example, a 32-expert transformer may acquire $10\times$ less knowledge in the same 100-exposure regime (mimicking rare knowledge) compared to dense models (Figure 11). Could Canon layers mitigate this due to their improved training dynamics?

Integrating Canon layers with MoE, however, poses a challenge. Canon-D relies on neighboring tokens' hidden states, conflicting with MoE's independent token-wise expert dispatching. Adapting Canon-D to MoE would require complex engineering. To avoid such complexity, we test Canon-ABC layers alone, which already significantly accelerate MoE knowledge acquisition and improve bit-per-parameter efficiency (Figure 11), recovering at least half of the MoE-induced capacity loss.

MLP with Squared ReLU. The Primer [57] paper proposes using ReLU^2 as the activation function in standard MLPs, reporting improved performance over gated MLPs (e.g., SwiGLU) on real-world data. They also claim this gain exceeds that of Canon-B(no-res), which they refer

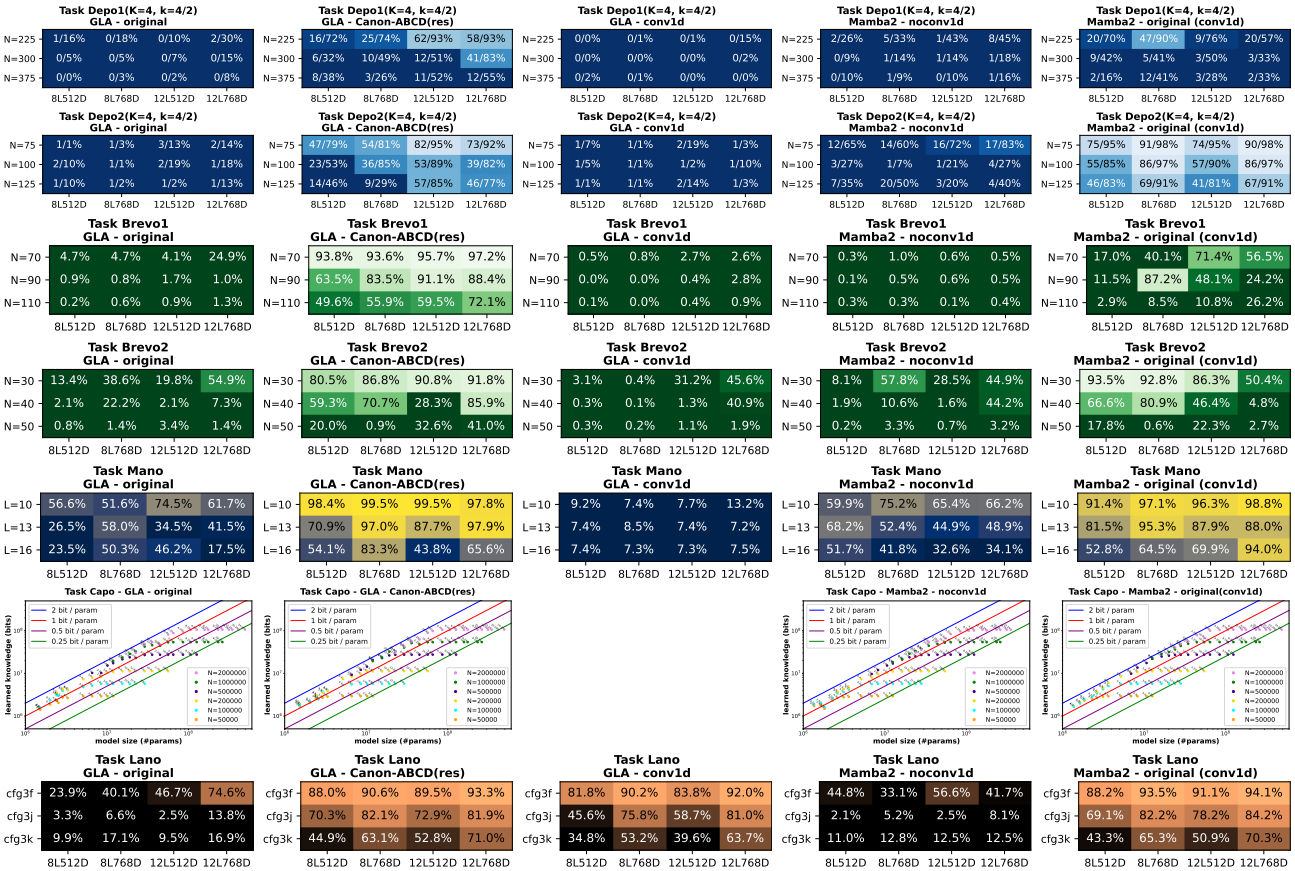


Figure 12: **Columns 1, 2, 3, 5**: Canon drastically improves GLA, making it comparable to Mamba2 (Result 6+7). **Columns 1, 4, 5**: Removing `conv1d` reduces Mamba2's performance back to match GLA (Result 8). *Remark.* Synthetic results here predict similar trends in real-life experiments (Result 12 and Figure 17).

to as "Multi-DConv-Head Attention." In our synthetic playground (see Figure 21), we confirm that ReLU^2 slightly improves standard MLPs (though not necessarily outperforming gated MLPs, consistent with recent findings [77]), while applying ReLU^2 to gated MLPs degrades performance. However, these effects are negligible compared to the gains provided by Canon layers.

Result 5 (Figure 11+20+21)**.** *Key insights for MLP and MoE architectures:*

- *Gated MLP slightly outperforms standard MLP (especially on* Mano*).*
- *Gated MLP reduces knowledge capacity (*Capo*); Canon layers partially recover this loss.*
- *ReLU^2 activation slightly improves standard MLP but degrades performance in gated MLP.*
- *Canon-ABC substantially improves MoE knowledge acquisition and bit-per-param capacity.*

6 When Linear Attention Meets Canon

Linear attention models reduce compute by compressing sequences into fixed-length representations, gaining popularity for their scalability and efficient handling of long contexts. We examine Canon layers integrated into Gated Linear Attention (GLA) [68]. GLA, like most linear attention models, compresses past tokens via a (gated) averaging mechanism. While efficient, gated averaging often diminishes the influence of nearby tokens—crucial for nearly all tasks. Canon layers explicitly introduce horizontal localized context flow, addressing this limitation and enhancing reasoning.

As shown in Figure 12, integrating Canon-ABCD layers substantially improves GLA performance across all benchmarks, transforming it from a weak baseline into a strong competitor. Despite its simpler design, GLA+Canon matches—and sometimes exceeds—Mamba2. Specifically, GLA+Canon significantly surpasses Mamba2 on reasoning breadth (BREVO1) and trails only slightly on knowledge manipulation. Our later Section 9 further confirms this upward trajectory in real-world academic-scale pretraining, across essentially all standard evaluation metrics.

Result 6 (Figure 12)**.** *Adding Canon layers:*

- *Skyrockets performance: GLA dramatically improves, increasing reasoning depth from 1-hop to 4-hop, doubling reasoning breadth, and gaining over 2× in knowledge manipulation length.*
- *GLA matches or surpasses Mamba2: Despite GLA's simpler design, these enhancements bring it on par or better compared to Mamba2, significantly surpassing it on task* BREVO1*.*

Following GLA's original publication, its authors introduced a `conv1d`-based enhancement — corresponding roughly to a partial Canon-B implementation with activation but without residual connections. Our ablation study (Figure 12+23) reveals minimal benefit of this variant: it yields no improvement on reasoning tasks (DEPO, BREVO), slightly worsens knowledge manipulation (MANO), and only moderately improves structural reasoning (LANO). By contrast, the full Canon-ABCD configuration consistently and significantly enhances performance on all these tasks. For clarity, when Canon layers are enabled in GLA, its built-in `conv1d` is disabled.

This highlights that merely altering attention projections (i.e. Canon-B alone) is insufficient. Instead, fully integrating localized composition across all sub-layers (Canon-ABCD) is essential for unlocking GLA's potential. To further confirm robustness, we evaluated a variant of GLA with a $1+\mathrm{elu}(x)$ feature map instead of the identity map. Canon layers produced similarly strong improvements here, illustrating their broad, architecture-agnostic applicability across different feature-map choices in linear attention models.

Result 7 (Figure 12+23)**.** *Residual links are essential for Canon layers in GLA. Partial Canon-B implementations (e.g.,* `conv1d` *without residuals) provide limited gains. Full Canon-ABCD with residuals can unlock GLA's performance across all tasks, even with non-linear feature maps.*

7 When Mamba Meets Canon

While Mamba2 is widely recognized as a state-space model (SSM), it quietly incorporates an auxiliary non-linear `conv1d` operation in each SSM block.[17] This `conv1d` mechanism, first introduced in the H3 model [23] as *shift-SSM*, effectively acts as a partial Canon-B layer: it performs horizontal mixing on selected coordinates, includes non-linear activation, and omits residual connections.

Surprisingly, this built-in `conv1d` plays a central role in Mamba2's performance, often surpassing the impact of its SSM design. Disabling it sharply degrades performance, reducing Mamba2 to levels comparable to GLA on both synthetic tasks (see Figure 12 for Mamba2, Figure 13 for Mamba2(mlp)) and real-world datasets (see Section 9). This raises a question: is Mamba2's strength primarily due to the Canon-like `conv1d` rather than its state-space formulation?

To isolate the contributions clearly, we replaced Mamba2's `conv1d` layer with Canon layers:

- For Mamba2, we use Canon-AB as the full-score configuration: Canon-A is placed before the SSM block with dimension $m = d$, and Canon-B placed within the SSM block with dimension

[17] Mamba1 includes this as well, but since Mamba2 consistently outperforms it, we only report Mamba2 results.

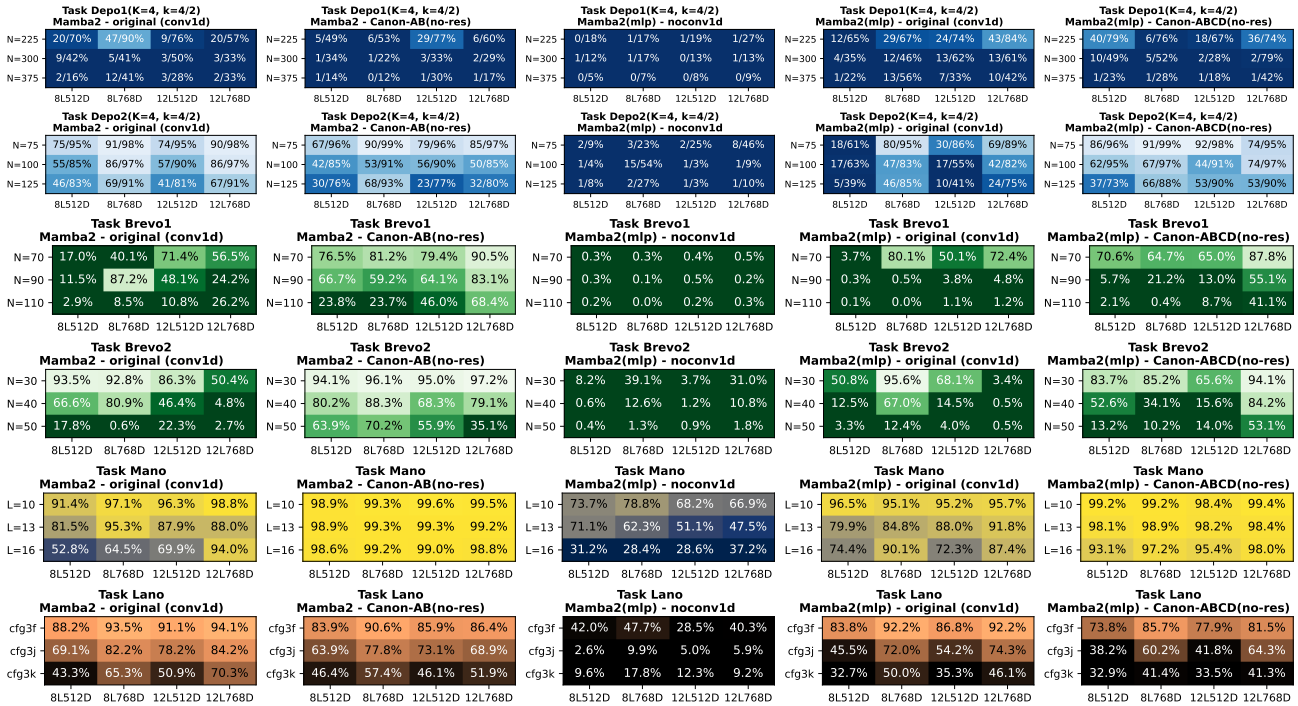


Figure 13: Mamba and Mamba(mlp) architectures without `conv1d`, with `conv1d` (original), and with full-score Canon. *Note:* Mamba - noconv1d and Mamba(mlp) - noconv1d perform similarly poorly (Figure 12).

$m = 4d + 144$.[18] In contrast, Mamba2's original `conv1d` applies only to a subset of $2d + o(d)$ dimensions with non-linear activation.

- For Mamba2(mlp), which alternates SSM blocks and gated MLP layers, we introduce Canon-C and Canon-D before and within the MLP sub-layers, respectively, together forming the full Canon-ABCD configuration.

As demonstrated in Figure 13, fully replacing the original `conv1d` with Canon layers consistently improves Mamba2, particularly on knowledge manipulation (Mano) and reasoning breadth (Brevo), surpassing even its original implementation.

Comparing Figure 12 with Figure 13, we conclude that the majority of Mamba2's performance boost comes from Canon-style horizontal information flow (either via original conv1d or our full Canon layers), not from its state-space formulation itself. Introducing Canon layers provides additional incremental gains, enabling Mamba2 to slightly exceed GLA—but the intrinsic state-space mechanism contributes comparatively limited improvement.

Result 8 (Figure 12+13)**.** *Key observations on Mamba2:*

- *Mamba2 contains an internal non-linear* `conv1d` *mechanism (a partial Canon-B) within its SSM blocks, which contributes more substantially to performance than its state-space design. Removing it greatly reduces performance, limiting Mamba2 to GLA-level capabilities.*
- *Replacing the internal* `conv1d` *with full Canon layers further improves performance, particularly in knowledge manipulation (*Mano*) and reasoning breadth (*Brevo*).*

(Also holds for Mamba1 [26]; in our playground Mamba1 is consistently outperformed by Mamba2.)

[18] The additive 144 arises from the Mamba2 specifications: `ssm_state_size = 64` and `num_heads = 16`. We also tested `num_heads = 8` and the results only went worse.

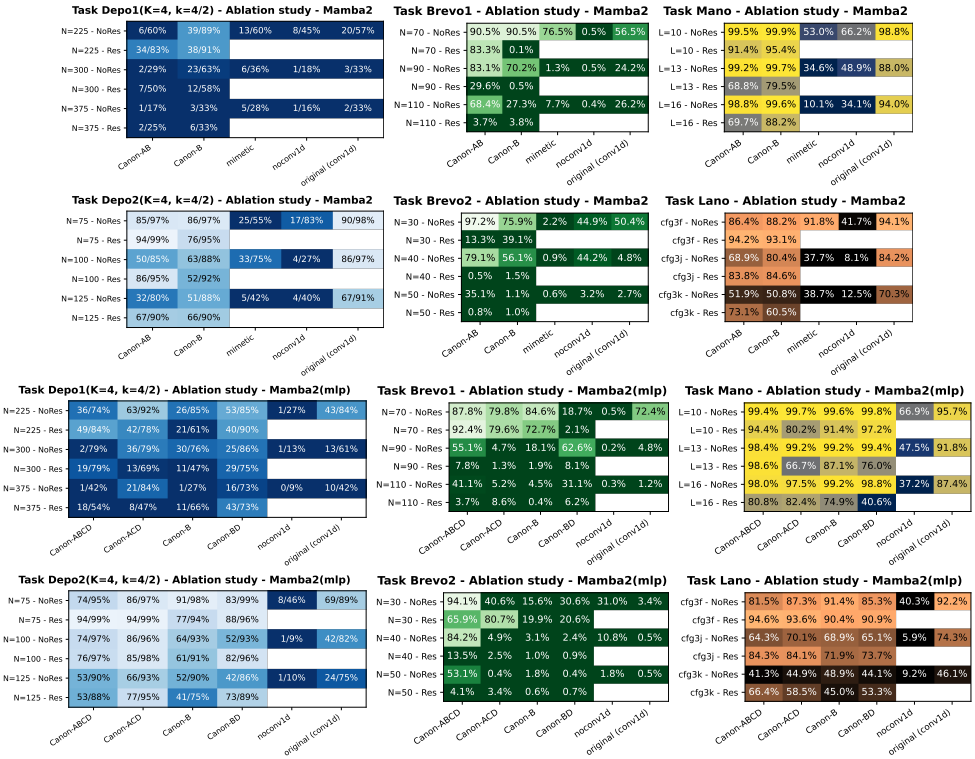


Figure 14: Ablation study of Mamba2 models of 12L768D size with Canon layers, Canon residuals, original non-linear `conv1d`, mimetic initialization. Full ablation studies (with additional model sizes, such as the effectiveness of Canon-ACD) are in Figure 29-30 of Appendix D.2.

7.1 Ablation Studies with Canon Layers

To better understand how Canon layers interact with Mamba, we performed ablation studies varying the number of sub-layers, residual connections, and initialization schemes.

Residuals on Canon. Residual links on Canon layers exhibited mixed effects. As shown in Figure 14, enabling residual connections improved hierarchical structural learning (Lano) and in-context reasoning depth (Depo), while disabling residuals benefited knowledge manipulation (Mano) and reasoning breadth (Brevo). While selective application of residual links across sub-layer configurations could optimize performance further, this is beyond the scope of this paper due to our primary focus on Transformer architectures.

Given Mamba's significantly weaker recursive reasoning (Depo, Lano) compared to Transformers, but stronger shallow, knowledge-driven reasoning (Mano, Brevo), we recommend disabling residual links for Canon layers with Mamba. This configuration better aligns with Mamba's core strengths. For hybrid architectures combining Mamba with Transformer layers (e.g., Samba [48]), we recommend optimizing Mamba for these shallow reasoning tasks, while delegating deeper, recursive reasoning (Depo, Lano) to Transformers.

Canon sub-layers. Adding more Canon sub-layers in Mamba2 provided improvements, though less pronounced than in Transformers. Notably, Mamba2(mlp) with Canon-ACD consistently outperformed both original Mamba2(mlp) and the Canon-B-only version. This confirms that Canon layers need not be tied to integration within the SSM block (as in original Mamba) and can also deliver benefits externally, similar to Transformer setups.

Mimetic initialization. We tested the mimetic initialization scheme proposed for Mamba [63], previously shown to enhance associative recall and selective copying, thereby improving length

Figure 15: Final comparison of base architectures equipped with full-score Canon layers: RoPE(♩), NoPE, Mamba2, and GLA. While Canon layers significantly enhance reasoning across all architectures (c.f. Figure 4), they are not a panacea. With Canon layers added, Mamba2 and GLA still underperform Transformers by 2× in reasoning depth, with results shown only for DEPO(K=4) due to zeros at K=8/16 (see Figure 29).

generalization. However, our ablation studies (Figure 14; see further details in Figure 29) found it offered no measurable benefit—and sometimes even degraded performance on other tasks. This suggests mimetic initialization may overfit length generalization objectives at the expense of performance on everything else. These results highlight the *importance* of evaluating architectural choices on a *diverse* task playground.

Result 9 (Figure 14)**.** *Canon layers are lightweight, versatile improvements for Mamba2, consistently enhancing reasoning and knowledge manipulation. Key findings include:*

- *Disabling Canon residuals improves* MANO *and* BREVO*, aligning with Mamba2's strengths; enabling residuals benefits recursive reasoning (*LANO*,* DEPO*), though often unnecessary.*
- *Canon layers remain effective when placed outside the SSM block, highlighting their generality as horizontal information-flow components independent of architectures.*
- *Mamba's mimetic init [63], designed for length generalization, harms shorter-context performance, highlighting the importance of architectural design using a* diverse *playground.*

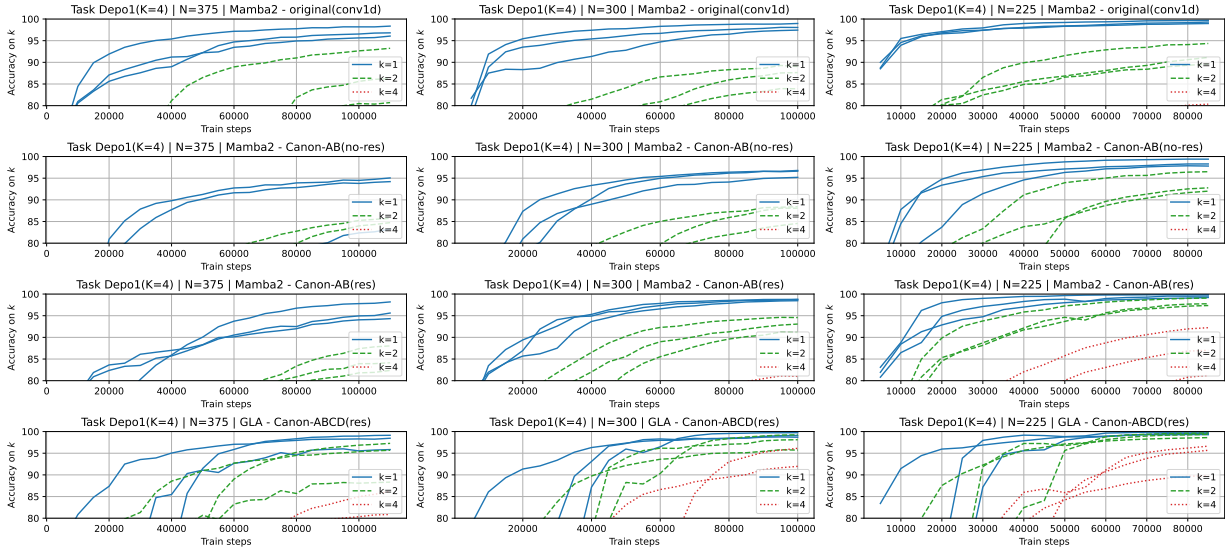


Figure 16: Training curves for 12L768D architectures on Task DEPO1(K=4), evaluated at $k = 1, 2, 4$ and $n = N$, with results shown across three learning rates for each k.

8 Final Comparison Across Base Models

Applying Canon uniformly to all architectures creates a controlled environment—like dropping them from the same height at the tower of Pisa—exposing architectural trade-offs. We exclude hybrid models (e.g., Griffin [20], Samba [48]) to isolate trade-offs among the core architectures.

Result 10 (Figure 15)**.** *With full-score Canon layers added, we find:*

- *reasoning depth: RoPE(♩) ≈ NoPE ≫ Mamba2 ≈ GLA (e.g., 2× deeper reasoning);*
- *reasoning breadth: RoPE(♩) ≥ NoPE ≈ Mamba2 ≈ GLA;*
- *knowledge capacity: Mamba2 ≈ GLA > RoPE(♩) ≈ NoPE;*
- *knowledge manipulation: Mamba2 ≥ RoPE(♩) > NoPE > GLA;*
- *hierarchical structure: RoPE(♩) > NoPE ≈ Mamba2 ≈ GLA.*

Remark 8.1. The initial comparison (Figure 4) does not represent a controlled comparison. In the original setup, Mamba2 includes non-linear conv1d layers (a subcase of Canon-B) while GLA and Transformers do not. By adding full-score Canon, the comparison becomes scientifically meaningful.

Challenges in Deep Reasoning for Linear-Time Models. Among all comparisons, reasoning depth remains the weakest point of Mamba and GLA, and potentially most linear-time models on the market. Due to their compression of in-context knowledge, these models struggle to reliably achieve 99% accuracy on even 1- or 2-hop information retrieval tasks (as demonstrated in Figure 16), despite prolonged training. When reasoning depth increases beyond 2 hops, errors from earlier steps accumulate significantly, leading to an overall inability to perform deeper reasoning tasks.

We emphasize: the inability to achieve high accuracy on 1- or 2-hop tasks is *not caused by insufficient recurrent memory*. Consider Mamba2: each layer passes $128d$ parameters (= expansion × ssm_state_size × hidden size d), sufficient to store the entire input sequence.[19] Additionally, Mamba2 performs well on simpler 1-hop tasks ($K = 1$) with even a single-layer network. Thus, the

[19] For example, in Task DEPO, representing N key-value pairs with a vocabulary size V requires at most $2N \log_2(V)$ bits. In DEPO2, we use $N = 75$ and $V \leq 2500$. This knowledge can easily fit within the memory state of Mamba2.

bottleneck is not information-theoretic. (Later real-life experiments shall also confirm this.)

The core limitation instead lies in the memory dynamics: how efficiently in-context information is encoded during compression and how reliably it is retrieved for reasoning. Inaccuracies in encoding or retrieval accumulate over hops, affecting performance in deeper reasoning tasks.

Thus, our findings pinpoint the *Achilles' Heel* of linear models for focused future research toward more intelligent linear designs. Until linear architectures overcome these limitations, hybrid approaches combining sliding-window attention (for deep reasoning) and linear or state-space components (for compressing longer contexts) remain the best practical solutions. Integrating Canon layers in both components can further maximize the overall performance.

Result 11 (Figure 16)**.** *Linear models like Mamba and GLA struggle with deep reasoning due to error accumulated from inefficient compression and retrieval, despite sufficient memory. Combining Transformers with linear models, both augmented with Canon layers, offers a compromise.*

9 Real-Life Experiments

We present real-life experiments at the academic scale, pretraining 1.3B-parameter language models on 100B tokens using the FineWeb-Edu [41] and SlimPajama [58] datasets, with a context length of 4096 (details in Appendix B). This setup mirrors configurations widely adopted in recent academic studies, including Titans [9], GatedDeltaNet (with `conv1d`) [69], and MTA [25], and reflects one of the most popular pretraining paradigms at this scale.

Model evaluation is conducted using at least two benchmark suites. The first suite is based on `lm-evaluation-harness` [24], as employed in prior studies [9, 69]. We focus on commonsense reasoning tasks, including PIQA [11], HellaSwag [75], WinoGrande [49], ARC-easy (ARC-e) and ARC-challenge (ARC-c) [17], SIQA [50], BoolQ [16], WikiText, and LAMBADA (LMB) [40]. These tasks evaluate sequence-to-sequence performance, where models solve multi-choice questions by scoring each option using log-likelihood. This set is referred to as *discriminative evaluation tasks*, and we adopt the original evaluation pipeline for consistency.[20]

The second suite follows the *Just Read Twice* (JRT) protocol [8], designed specifically to improve the reliability of generative evaluations for smaller-scale pretrained models.[21] The six tasks in this suite include SWDE and FDA (originally described in [8]), as well as SQuAD(v2) [47], TriviaQA [33], Natural Questions (NQ) [37], and DROP [21]. JRT-enhanced variants are denoted as FDA2, SWDE2, SQuAD2, etc., to indicate prompt augmentation. This suite is referred to as *generative evaluation tasks*, and we again adopt the original evaluation codebase for comparison.

Key Observations. Despite identical training setups, model performance varies significantly due to random seeds, which affect initialization and data sampling order. For example, Llama(RoPE)-1.3B exhibits **fluctuations exceeding 4% on LAMBADA, 3% on BoolQ**, and 1–3% on other discriminative tasks. Variability is greater for generative tasks: **FDA fluctuates by 9%, SWDE by 8%**, and other generative tasks by 3–5%—even with JRT prompts enabled. *Hence, only differences exceeding these thresholds are considered statistically meaningful.* From Figure 17:

- *Generative evaluation tasks:* Linear models (Mamba2, GLA, and GatedDeltaNet) substan-

[20] Following tradition [9, 68, 69], we use (`acc_n`) for HellaSwag and ARC-c, but `acc_n` for other tasks.

[21] Generative testing can be noisy at this scale, as such models often struggle with prompt comprehension. JRT addresses this by repeating the context and question twice, allowing models to more accurately reveal their intrinsic generative capabilities.

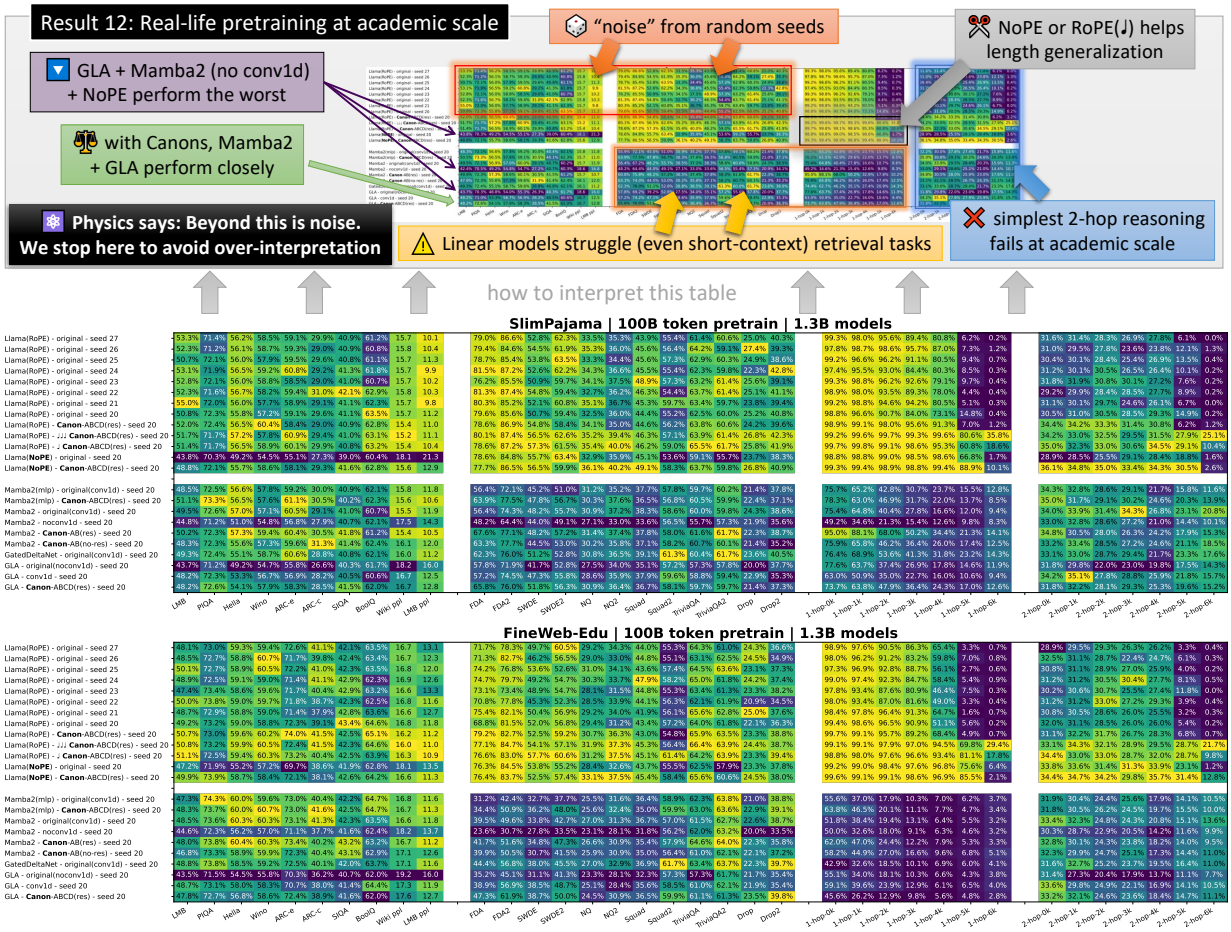


Figure 17: Performance of 1.3B models pretrained on 100B tokens, evaluated on discriminative (left), generative (middle), and 1/2-hop reasoning (right) tasks. Best of 2 learning rates for Llama; 3 for GLA, Mamba, and GatedDeltaNet. For GPT2 variants (e.g., standard MLP, ReLU^2), see Figure 24.

tially underperform full Transformers, even on contexts shorter than their training length.[22] Retrieval-heavy tasks like FDA/SWDE amplify this gap, aligning with our findings in Result 11.

- Both GLA and NoPE show poor performance in their base configurations, but significant improve with full-score Canon layers. GLA+Canon often matches or surpasses Mamba2, while NoPE+Canon performs comparably to RoPE across most tasks. These confirm Result 3+6. Conversely, removing `conv1d` makes Mamba2 perform as bad as GLA, confirming Result 8.
- While RoPE, RoPE+Canon, and NoPE+Canon excel across tasks without notable differences, linear models like GLA+Canon, Mamba2+Canon (or `conv1d`), and GatedDeltaNet (with `conv1d`) perform similarly. This suggests that at the academic scale (1.3B parameters and 100B tokens), pretraining does not reliably highlight finer architectural differences.

Babilong and Multi-Hop Reasoning. The Babilong dataset [36] embeds the original bAbi dataset [66] into passages with varied lengths of "junk" information to test multi-hop reasoning in long contexts. At the academic pretraining scale, Babilong is overly challenging.[23] Our Figure 25 in

[22] Generative task prompts are capped at 1024 tokens (or 2048 tokens for JRT prompts) based on the original codebase, while models were trained with 4096-token sequences.

[23] For example, in `babilong.qa2`, the passage "Charlie went to the kitchen. Charlie got a bottle. Charlie moved to the balcony." is followed by the 2-hop question: "Where is the bottle?" Models fail even without junk context, achieving below 37% accuracy—essentially random guessing

the appendix show that it is difficult to use Babilong to evaluate reasoning or length generalization. The only statistical significant findings (in our standard) are:

- Linear models underperform Transformers *even in short contexts*, confirming their weakness stems *not* from memory but from *inefficiencies in compression and retrieval* (c.f. Result 11).
- Transformers show significant long-context gains when RoPE is reduced (RoPE♩) or removed (NoPE), particularly in junk contexts extending to 4k tokens in length.

To address Babilong's limitations, we introduce simpler evaluation tasks for multi-hop reasoning. Task 1-hop-L embeds five birth-year statements within Wikipedia passages of length L, prompting models to recall birth years. Task 2-hop-L uses three birth-year statements followed by three equivalence links (e.g., "X was born the same year as Y"), prompting models to infer the birth years of linked names. Details are in Appendix B. These tasks are designed to be the most natural, leveraging common English (e.g., birth years). Results from Figure 17 show:

- All models struggle with 2-hop-L, only 30–36% accuracy (i.e., random guess) even with $L = 0$.
- 1-hop-L distinguishes architectures: full Transformers outperform linear models at $L = 0$, while NoPE-based Transformers and RoPE(♩) length-generalizes better as L increases.

Result 12 (Figure 17+25)**.** *Academic-scale pretraining—1.3B-parameter models trained on 100B tokens with a context length of 4096—exhibits high noise and limited resolution, with most architectural differences statistically insignificant. Yet, some consistent findings emerge:*

- *Linear models (Mamba2, GLA, GatedDeltaNet) underperform full Transformers even on short-context retrieval tasks (1-hop-L, FDA, SWDE), even with Canon layers (c.f. Result 11).*
- *Canon layers strongly elevate GLA to Mamba2-level, NoPE to RoPE-level (c.f. Result 3+6); conversely, removing* `conv1d` *sharply downgrades Mamba2 to GLA (c.f. Result 8).*
- *All models struggle with 2-hop tasks, even in 100-token contexts, highlighting the limits of academic-scale pretraining.*
- *Reducing or removing RoPE (i.e., NoPE or RoPE♩) improves long-context generalization.*

10 Conclusion and Future Direction

Academic-scale pretraining suffers from high noise and failed multi-hop reasoning, hindering reliable architectural comparison. Our controlled synthetic playground offers a **cost-effective, principled alternative**: by decomposing intelligence into atomic tasks, we discover and optimize *Canon layers*—lightweight constructs that enhance reasoning depth and breadth, knowledge capacity and manipulation, and structural reasoning across diverse architectures.

Canon layers revive weaker models (e.g., NoPE, GLA) to match or surpass stronger baselines (e.g., RoPE, Mamba2), reduce reliance on RoPE to improve length generalization, and pinpoint that linear models' depth limitations arise from compression/retrieval inefficiencies rather than memory. Like residual connections or LoRA—simple yet powerful—Canon layers may become a minimal yet broadly applicable architectural primitive.

While our academic-scale real-world experiments align with synthetic findings, industrial-scale validation remains crucial; we hope our systematic, economical methodology **encourages future investigations** at larger scales. We plan to open-source our playground and evaluation suite to support rigorous, reproducible architecture research.

Future Directions. Several interesting directions arise from this work:

- Alternative Canon Implementations. We focused on simple linear convolutional (kernel size 3) Canon layers for their simplicity and efficient CUDA kernels. Future work should explore dynamic, adaptive convolutions—with weights conditioned on hidden states to enable gating—to assess whether performance gains justify the added computational overhead.
- Fine-grained Canon Design. We briefly explored selective application (e.g., early layers) and cross-layer connections—e.g., $h'\ell + 1 = h\ell + 1 + \text{Canon}(h_\ell)$—which can fuse multiple intra-layer Canon operations into a single step, improving efficiency. A systematic evaluation within our synthetic framework could identify optimal Canon configurations. We are open to exploring this direction further, especially if the community expresses significant interest.
- Evaluating Emergent Architectures. We selected one representative per architecture family to ensure controlled comparisons and consistent inclusion of Canon layers. Without this rigor, results may misleadingly attribute Canon's gains to inherent architectural differences (e.g., Mamba2's built-in conv1d). With controlled comparisons in mind, future work can fairly evaluate emergent architectures, potentially discovering new components with statistically significant improvements.
- Enriching the Synthetic Playground. Our five synthetic tasks are only a starting point. Designing additional tasks that isolate other architectural capabilities *beyond those revealed here*—while remaining as atomic as possible—is crucial for finer-grained characterization of model strengths and weaknesses.
- Interpretability and Probing. We omitted interpretability and probing analyses here for clarity, despite existing frameworks for most tasks (e.g., Lano [3], Capo [5], Mano [6], Brevo [71, 72]). We have conducted preliminary probing for Depo, revealing internal model strategies such as positional parsing (even/odd positioning encoding "$\to a$" or "$a \to$") and preprocessing of permutations before the first query (analogous to Brevo [71]). We choose not to include them for clarity, as this paper focuses on architectural comparison.
- Sparking New Architecture Designs. By pinpointing specific weaknesses (e.g., linear models' reasoning depth limits and compression inefficiencies), our framework provides targeted signals for improved future designs. We hope synthetic benchmarking informs and inspires the next generation of architecture innovations.

Appendix

This appendix contains full technical specifications and implementation details for all experiments presented in the main paper. It is intended to support reproduction and in-depth inspection. We provide complete training protocols and evaluation procedures for all five synthetic tasks (Depo, Brevo, Capo, Mano, Lano), real-life experiments (1-hop-L, 2-hop-L, Babilong), and 100B-token SlimPajama/FineWeb-Edu pretraining. We also document the architectural configurations for all models, including Transformers, GLA, Mamba variants, and MoEs. Additional ablation figures, KL-divergence evaluations, and variant comparisons are included for readers interested in deeper technical insights or replication of results.

A Details on Synthetic Pretraining Tasks

We intend to release the code for generating all synthetic pretraining datasets used in this paper, though this may require additional time. To make this paper fully self-contained, we provide detailed specifications below.

Remark A.1. Throughout this paper, we utilize combinations of A100, H100, and H200 GPUs with bf16 mixed-precision training. While we report the total batch size used in our experiments, we do not specify the exact number of GPUs, as this does not materially affect the results.[24]

A.1 Details on Task Depo: Mental Reasoning Depth

The synthetic pretraining task Depo is designed to evaluate mental reasoning depth by requiring multi-step traversal over directed permutations. The dataset is defined by two parameters: the maximum permutation size N and the reasoning depth K. Each problem instance is generated as follows:

First, a permutation length n is sampled uniformly from $\{3, 4, \ldots, N\}$. A directed permutation of n nodes is then created, representing a cycle where each node points to its successor: $x_1 \to x_2 \to \cdots \to x_n \to x_1$. The permutation is presented as edges in the form of ordered pairs (x_i, x_{i+1}), but these edges are shuffled randomly into a sequence of $2n$ tokens. This random ordering ensures that the original cycle structure is not immediately apparent, which would otherwise make the task trivial. The final data format is:

```
<bos> x1 y1 x2 y2 ... xn yn <query k_1> q_1 <ans> a_1 ... <query k_t> q_t <ans> a_t
```

Here, $x_i \to y_i$ represents shuffled edges of the permutation. For each query q_j, a node is randomly chosen from $\{x_1, \ldots, x_n\}$, and its k_j-th successor in the permutation is computed based on the reasoning depth $k_j \in [K]$, sampled uniformly. The correct answer a_j is the k_j-th successor of node q_j. The number of queries t is set as $\min(10, n)$ to balance computational feasibility while ensuring smaller graphs remain interpretable.

Two variants of Depo are used:

- Depo1: Each node name is encoded as 1–2 tokens, with a vocabulary size of 50.

[24] For instance, training with a single GPU and a batch size of 128 is equivalent to training with 64 GPUs where each GPU processes a batch size of 2. Our codebase supports dynamic GPU allocation, ensuring the total batch size is fixed across training runs while the number and type of GPUs may vary.

- DEPO2: Each node name spans 5–7 tokens using a small vocabulary size of 4, introducing ambiguity that challenges the model's disambiguation capabilities.[25]

In addition to node names, special tokens are used: `<bos>`, `<ans>`, and `<query k>` for $k \in \{1, \ldots, K\}$. The total number of special tokens is $K + 2$.

Sampling distribution. To ensure controlled task difficulty progression, n is sampled proportionally to $\frac{1}{\sqrt{N}+n}$. This distribution biases training toward simpler cases early on, allowing the model to gradually build foundational reasoning skills before encountering harder examples. Although this distribution is not perfect, it is both simple and effective, enabling clean comparisons between architectural designs without introducing unnecessary hyperparameter complexity. More sophisticated curriculum-based approaches, such as scheduled difficulty [38], may provide an alternative solution but could introduce significant noise, thereby complicating controlled comparisons.

Remark A.2. This distribution was proposed and tested thoroughly by ZA in 2023 in a number of settings, and subsequently tested (via private communication) by Alfarano in modular arithmetic pretraining [51], where it was benchmarked against other options and shown to also perform well. While synthetic data like this cannot fully replicate the intricacies of real-world distributions, it allows us to simulate an ideal training regime. This forward-looking approach anticipates future improvements in pretraining data—such as higher-quality datasets or RL-based post-training—and evaluates model architectures based on their scalability under such optimal conditions.

Training protocol. To reduce computational cost, we employ label masking: cross-entropy loss is computed only on tokens associated with `<ans>` and a_j. This optimization halves training duration without affecting architectural comparisons. Problem instances are generated online, concatenated, and aligned into 2048-token context windows. Left alignment ensures that the first problem instance in each context is never truncated, as truncation leads to incomplete edges and unusable data.

Evaluation protocol. During evaluation, the permutation size is fixed at $n = N$, and reasoning depth is tested at both $k = K$ (maximum depth) and $k = K/2$ (intermediate depth). The protocol mirrors training by generating and concatenating evaluation samples online into 2048-token windows. Accuracy is reported over all answer tokens in the window, ensuring that results are stable regardless of whether answers appear early or late in the sequence.

Data splits and hyperparameters. For DEPO1, we use $N = 375$, 300, 225 and primarily $K = 8$, while testing $K = 4$ for weaker models. Models are trained from scratch with fresh data while using a fixed random seed to ensure data consistency across architectures. Training uses a batch size of 128, AdamW optimizer ($\beta = 0.9, 0.98$ and $\varepsilon = 10^{-6}$), weight decay of 0.03, learning rate warmup for the first 1000 steps, and cosine decay to 10%. Training steps are set to 112.5k, 100k, or 87.5k, adjusted for the problem lengths $N = 375$, 300, 225. The best accuracy is reported across four runs using learning rates $\{0.0003, 0.0005, 0.001, 0.002\}$.

Similarly, in DEPO2, we use $N = 125$, 100, 75 and $K = 16$ (or $K = 4$ for weaker models). Training steps are set to 150k, 125k, and 100k, respectively.

A.2 Details on Task Brevo: Mental Reasoning Breadth

Our pretraining synthetic task BREVO is designed to test mental reasoning breadth by requiring a subgraph topological sort from a given directed acyclic graph (DAG). The dataset is defined by a maximum graph (node) size N. For each problem instance, we first sample a graph of size

[25] Multi-token names are generated such that the first $\ell - 1$ tokens are chosen from $[1, V]$, while the final token is selected from $[V + 1, 2V]$. This creates implicit word boundaries similar to those handled by BPE-based tokenization strategies, such as GPT2Tokenizer.

$n \in \{3, 4, \ldots, N\}$ using the same sampling distribution $\propto \frac{1}{\sqrt{N}+n}$ as employed in DEPO, and generate data in the following format:

```
<bos> x1 y1 x2 y2 ... xm ym <query> q <ans> a1 a2 ... ap <eos>
```

Here, the $2m$ tokens define m directed edges $x_i \to y_i$ spanning n nodes, meaning that y_i depends on x_i. Given a query vertex q, the model must return all vertices it recursively depends on, in topological order starting from the leaves. Specifically, if $u \to v \to q$, the model must output u before v.

DAG generation protocol. After sampling n, we generate the random DAG as follows. First, we randomly shuffle all the vertices and begin inserting edges. We select a random number $L \in \{1, \ldots, \lceil \frac{n-1}{4} \rceil + 1\}$, designating the first L vertices as leaves (no incoming edges). Starting from vertex $L + 1$, we iteratively process each vertex by selecting all preceding vertices that have an out-degree of at most 3. From this set, we randomly pick a subset of between 1 and up to 4 vertices and connect them to the current vertex. This process continues until all vertices are traversed, yielding a DAG with a maximum in-degree and out-degree of 4.[26]

At this point, the vertices naturally form a topological order from left to right. We then select a random query vertex from the last quarter of the vertices. Choosing vertices closer to the right increases the depth of the dependency graph while avoiding degenerate cases where all nodes are reachable (such as if the query were the last vertex). Finally, we reshuffle all the vertices and assign random names to them. Vertex names are uniquely selected, as described below.

Vertex names. In BREVO1, each vertex name consists of a single unique token, randomly selected from $\{1, \ldots, N\}$. In BREVO2, each vertex name spans 2–4 tokens using a vocabulary of size 4, which introduces ambiguity (e.g., multiple token combinations can encode unique vertex names). See Footnote 25 for the method used to generate multi-token words. Aside from vertex names, we use 4 distinct special tokens: `<bos>`, `<query>`, `<ans>`, and `<eos>`.

Training protocol. To reduce computational costs, we enable label masking, where the cross-entropy loss is computed only on `<ans>`, `<eos>`, and a_j tokens. Selective testing showed that this technique saves training time without affecting architectural comparisons. Instances are generated online, concatenated, and left-aligned into context windows. By left-aligned, we mean that the first instance in each context window is never truncated. Without left alignment, truncation of the first instance would render it incomplete (e.g., missing edges in the graph), making the instance a useless training example.

Evaluation protocol. During evaluation, we fix $n = N$ and test only the largest graph. The model is prompted with a random DAG of size n and query vertex q, and tasked to generate the answer sequence $a_1, \ldots, a_p$. The generated sequence is then parsed and validated against the following criteria:

- The answer sequence must contain all reachable vertices from q and no non-reachable vertices.
- The vertices in the answer sequence must appear in a valid topological order. Since topological orderings are not unique, any valid ordering is accepted.

Invalid tokens, duplicate outputs, or missing vertices are not accepted, and no partial credit is given.

Training details. In BREVO1, we use $N = 110, 90, 70$ with vertex names consisting of one token, and each problem fits within 1024 tokens. Models are trained from scratch with fresh data but

[26] Constraining the maximum in-degree and out-degree to 4 prevents the dependency graph from becoming too shallow, which would make the task trivial.

a fixed seed (ensuring pretraining data consistency across model architectures). Training uses a context length of 1024, a total batch size of 256, AdamW optimizer ($\beta = 0.9, 0.98$ and $\varepsilon = 10^{-6}$), weight decay of 0.03, learning rate warmup over the first 1000 steps, and cosine decay to 10%. Pretraining lasts 150k, 125k, or 100k steps respectively for $N = 110, 90, 70$, accounting for the varying problem lengths. We report the best performance out of four runs using learning rates $\{0.0003, 0.0005, 0.001, 0.002\}$.

In BREVO2, we use $N = 50, 40, 30$, with vertex names spanning 2–4 tokens, and each problem fits within 1536 tokens. Models are trained in the same manner as BREVO1, except that we use a context length of 1536, a total batch size of 192, and pretraining lasts 250k, 225k, or 200k steps respectively for $N = 50, 40, 30$.

The comparison between BREVO1 and BREVO2 demonstrates that the ambiguity introduced by multi-token vertex names does not noticeably impact architectural comparisons, which is the focus of this paper.

A.3 Details on Task Capo: Knowledge Capacity

The synthetic pretraining task CAPO borrows directly from Allen-Zhu and Li [7], where the authors introduced the $\mathsf{bioS}(N)$ dataset. This dataset contains N biographies of randomly generated individuals, each described by six attributes: birth date, birth city, university, major, employer, and working city.[27]

To represent these biographies in natural language, each individual is described via randomly selected English sentences for every *exposure* to the pretraining data. Sentence templates correspond to the individual's attributes, ensuring diverse paraphrasing across exposures. For example:

> Anya Briar Forger was born on October 2, 1996. She spent her early years in Princeton, NJ. She received mentorship and guidance from faculty members at Massachusetts Institute of Technology. She completed her education with a focus on Communications. She had a professional role at Meta Platforms. She was employed in Menlo Park, CA.

The diversity in writing ensures that models learn to store explicit knowledge about an individual's attributes, rather than merely memorizing surface-level patterns in specific templates [5, 6]. Following the recommendations of [7], we pretrain models over 100 *exposures* per individual, which provides a controlled environment for comparing architectural differences. Training beyond 100 exposures diminishes architectural differences, as longer training typically allows all models to converge toward similar levels of performance [7].

Knowledge format independence. Previous experimental evidence suggests that a model's knowledge capacity does not heavily depend on the specific format in which the knowledge is stored. For example, one could consider synthetic alternatives such as longer word lengths, different vocabulary sizes, or even abstract encoding formats. Importantly, any such synthetic configuration remains a reliable discriminator for comparing model architectures. For simplicity and interpretability, however, we adhere to the more English-like biography format in $\mathsf{bioS}(N)$, aligned with [7].

Clean experimental comparisons. Models could alternatively be pretrained on exposures distributed according to power-law dynamics or incorporating infrequent "junk data." While such approaches might better mimic real-life datasets, they introduce subtle stochastic effects that can depend heavily on the formatting of rare samples. To avoid confounding factors, we adopt the

[27] The working city is derived from the employer's headquarters, while all other attributes are sampled uniformly and independently. Possible attribute domains include $N_0 = 400 \times 400 \times 1000$ person names, $12 \times 28 \times 200$ birth dates, 200 birth cities, 300 universities, 100 majors, 263 employers, and two pronouns.

cleaner 100-exposure baseline for pretraining individual biographies, as it allows for clearer isolation of architectural capabilities.

Evaluation protocol. After pretraining on $\mathsf{bioS}(N)$ data, knowledge capacity is measured based on the number of *bits* a model reliably stores. This quantity is further normalized to **bits per parameter** to account for model scale. Partial correctness (e.g., recalling the year but not the full date of birth) is accounted for in the bit computation to ensure fine-grained evaluation of knowledge storage. For detailed computation, we direct readers to [7]. Unlike other tasks presented in this paper, measurement of bits per parameter requires varying both data sizes N and model sizes to compute the Pareto frontier of knowledge capacity versus parameter count. For this reason, we vary N between 50K and 2M while testing models ranging from 1M to 500M parameters.

Pretraining setup. To ensure consistency across all architectures, pretraining uses the GPT-2 tokenizer and ties weights for embedding and output layers. Tying weights ensures consistent learning dynamics across model families (e.g., GPT, Llama, Mamba, GLA), while limiting the vocabulary size to 3275 tokens (from GPT-2's original 50257 tokens), as the $\mathsf{bioS}(N)$ dataset does not utilize the entire vocabulary.

Batch size, learning rate decay, and other hyperparameters strictly follow the 100-exposure baseline outlined in [7], with only minor modifications. Specifically, we test two learning rates per configuration (selected from their optimal three) and report the best results. As a result, our reported knowledge capacity in Figure 4 may slightly deviate from the original results, though introducing Canon layers restores capacity without adding hyperparameter choices.

Hyperparameters for dense models. The following hyperparameters were used for dense models in the 100-exposure setup:

- For $N = 50$K: weight decay $wd = 0.01$, learning rates $lr = 0.001/0.0005$, batch size 12.
- For $N = 100$K: $wd = 0.01$, $lr = 0.001/0.0005$, batch size 24.
- For $N = 200$K: $wd = 0.01$, $lr = 0.001/0.0005$, batch size 48.
- For $N = 500$K: $wd = 0.01$, $lr = 0.001/0.0005$, batch size 96.
- For $N = 1$M: $wd = 0.01$, $lr = 0.001/0.0005$, batch size 192.
- For $N = 2$M: $wd = 0.01$, $lr = 0.0005/0.0003$, batch size 384.

Hyperparameters for MoE models. Mixture-of-Experts (MoE) training was conducted using the `tutel_moe` package [29], consistent with [7]. MoE training uses 32 experts with $topk = 1$ and $cap\_factor = 2$. Due to the larger learning rates required for MoE-based pretraining, we use the following hyperparameters:

- For $N = 50$K: $wd = 0.01$, $lr = 0.005/0.002/0.001$, batch size 12.
- For $N = 100$K: $wd = 0.01$, $lr = 0.005/0.002/0.001$, batch size 24.
- For $N = 200$K: $wd = 0.01$, $lr = 0.005/0.002/0.001$, batch size 48.
- For $N = 500$K: $wd = 0.01$, $lr = 0.002/0.001$, batch size 96.
- For $N = 1$M: $wd = 0.01$, $lr = 0.002/0.001$, batch size 192.
- For $N = 2$M: $wd = 0.01$, $lr = 0.001/0.0005$, batch size 384.

A.4 Details on Task Mano: Knowledge Manipulation

The synthetic pretraining task Mano evaluates a model's ability to manipulate stored knowledge mentally without relying on explicit intermediate cues (e.g., Chain-of-Thought reasoning). Unlike memorization tasks, Mano requires multi-step internal computation, testing the model's capacity for hierarchical manipulation.

Task format and setup. The dataset is defined by a maximum length L, with each instance consisting of arithmetic expressions of ℓ operations, where ℓ is sampled uniformly from $[1, L]$.

Expressions are presented in prefix (pre-order) notation to eliminate ambiguities in parentheses and operator precedence. For example, a length-$\ell = 3$ instance is:

```
<bos> <len_3> + * a b - c d <ans> ans
```

This corresponds to the expression $((a \times b) + (c - d)) \bmod 23$, where operands a, b, c, and d are integers sampled uniformly from $[0, 22]$. The task involves three operations (+, -, *), each represented as distinct tokens, with all computations performed modulo 23.

The factual base consists of three 23×23 arithmetic tables (addition, subtraction, and multiplication), which models learn implicitly during pretraining. Operands are encoded as single tokens from $[0, 22]$, while special tokens (<bos>, <ans>, and <query_ℓ> for $\ell \in [L]$) structure the sequence.

Expressions are generated recursively: the first operator is sampled uniformly from the three available options, and its operands are split into sub-lengths ℓ' and $\ell - 1 - \ell'$ (with ℓ' chosen uniformly), recursively generating sub-expressions.

Why modular arithmetic? Modular arithmetic (mod 23) ensures manageable knowledge size while introducing sufficient diversity in intermediate and final results. Similarly, limiting operations to addition, subtraction, and multiplication simplifies task design while retaining depth, enabling models to focus on hierarchical manipulation instead of memorizing surface-level patterns.

Training protocol. Models are pretrained on three datasets corresponding to difficulty levels $L = 16$, $L = 13$, and $L = 10$. The cross-entropy loss is applied over all tokens (problem description and answer), without label masking, since hierarchical manipulation requires attention across the full sequence. Instances are generated online, concatenated, and left-aligned into context windows of length 1024.

Models are trained from scratch using fixed random seeds for consistency across architectures. Training lasts 110k, 95k, and 80k steps for $L = 16$, $L = 13$, and $L = 10$, respectively. Hyperparameters include a batch size of 64, AdamW optimizer ($\beta = 0.9, 0.98$ and $\varepsilon = 10^{-6}$), weight decay of 0.1, learning rate warmup for 1000 steps, and cosine decay to 10% of the initial learning rate. Results are reported based on eight training runs with learning rates $\{0.0001, 0.0002, 0.0003, 0.0005\}$ and two seeds.

Evaluation protocol. During evaluation, expressions are sampled from the same distribution used for training, with ℓ fixed at L (maximum difficulty). Accuracy is computed over all problem instances within 1024-token context windows, including non-first instances. Since outputs are single tokens representing exact modular arithmetic results, partial correctness is not applied.

A.5 Details on Task Lano: Hierarchical Language Structure

The synthetic pretraining task Lano evaluates a language model's ability to perform structural reasoning, specifically long-range structural planning that requires dynamic programming to resolve ambiguity. Unlike in-context reasoning tasks (e.g., Depo, Brevo) or knowledge reasoning tasks (e.g., Mano), Lano challenges models to learn hierarchical structures governed by probabilistic context-free rules and process sequences that cannot be resolved locally.

Task format and setup. Sentences are generated probabilistically using context-free rules. The cfg3f dataset [7] starts with the root non-terminal (NT) symbol 22, which uniformly expands into one of four rules:

$$22 \mapsto 20\ 21, \quad 22 \mapsto 20\ 19\ 21, \quad 22 \mapsto 21\ 19\ 19, \quad 22 \mapsto 20\ 20.$$

Each rule is chosen with probability 1/4, ensuring uniform randomness. Rules are applied recursively and probabilistically to NT symbols (e.g., 19, 20, 21), replacing all NT symbols with

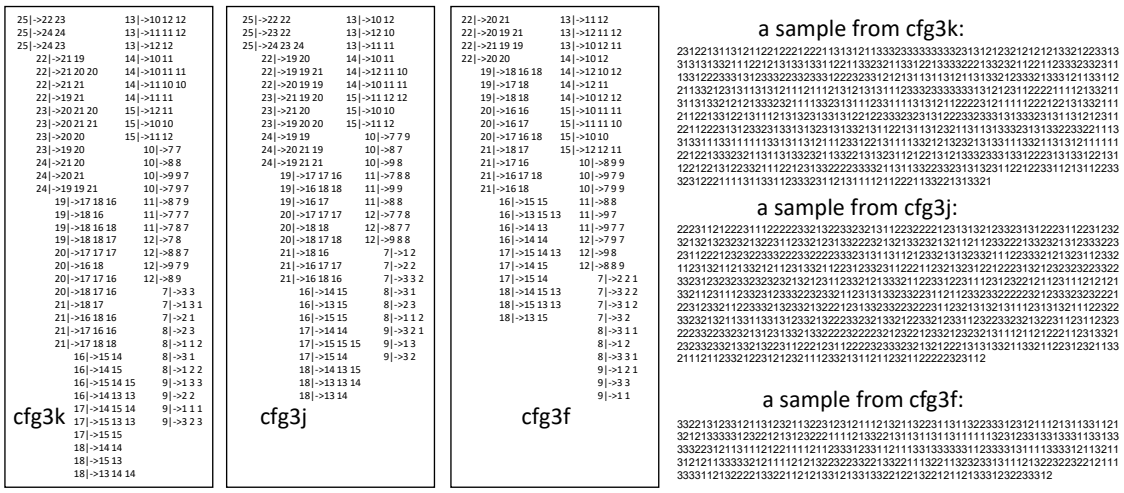


Figure 18: Task LANO: our constructed dataset against the cfg3f dataset from [3].

terminal (T) symbols 1, 2, or 3. The process generates sentences composed entirely of terminal symbols based on probabilistic expansions.

Pretraining involves predicting next tokens in CFG-generated sequences without access to the underlying rules, requiring models to learn structural reasoning implicitly. During evaluation, models are prompted with a single <bos> token and tasked to generate CFG-compliant sentences using temperature 1. Accuracy is assigned only for fully valid sentences, with no partial credit applied.

Parsing difficulty and ambiguity. Parsing CFG-generated sequences is uniquely challenging because resolving derivation chains requires global reasoning. For example, parsing "221213133" requires resolving structural ambiguity between terminal symbols that cannot be inferred from local patterns alone. Instead, parsing requires an $O(n^3)$ dynamic programming algorithm to globally reconstruct relationships across the sequence, even when CFG rules (from Figure 18) are explicitly available. During pretraining, models face additional difficulty as they must learn these relationships without direct access to the probabilistic rules.

Building upon cfg3f as a baseline, we introduce two extended datasets in this paper:

- cfg3k: Retains the probabilistic framework of cfg3f but increases depth by one level, doubling sequence length and increasing parsing complexity by eight times due to the cubic nature of dynamic programming ($O(n^3)$).
- cfg3j: Extends cfg3f by one level but reduces the number of rules, creating shorter sequences with intermediate difficulty between cfg3f and cfg3k.

Both datasets use the same probabilistic generation process and are detailed in Figure 18.

Training details. Pretraining uses cross-entropy loss computed over all tokens without label masking. Sentences are generated online, concatenated, and aligned into context windows. For cfg3f, we use a context length of 512 as in [7], while longer datasets cfg3j and cfg3k require extended context lengths of 1536.

Models are trained from scratch using fixed seeds for consistency across architectures. Training uses a batch size of 96, AdamW optimizer ($\beta = 0.9, 0.98$ and $\varepsilon = 10^{-6}$), weight decay of 0.1, no learning rate warmup, and linear decay to 0. Pretraining lasts 100k steps, and results are reported from four training runs using learning rates $\{0.0002, 0.0003, 0.0005, 0.001\}$.

Evaluation details. During evaluation, models generate sentences from a <bos> prompt using

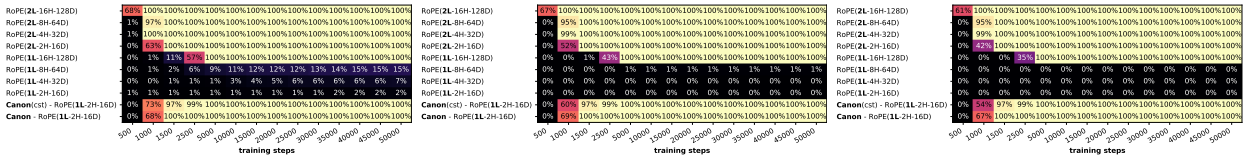
temperature 1 and beam width 1.[28] Generated sentences are validated using an $O(n^3m)$ dynamic programming parser (n: sequence length, m: CFG rules) to confirm compliance. An alternative evaluation computes KL divergence between the model's next-token prediction distribution and the ground-truth CFG predictions. Both methods yield consistent architecture comparisons.

B Details on Other Experiments

This section provides a brief description of additional tasks used in the paper.

Full Copy. In Figure 5, we evaluated the performance of models with 1 or 2 layers on a trivial pretraining task. This task involves choosing $N = 500$ and generating a sequence starting with `<bos>`, followed by a random permutation of N tokens between 1 and N, then appending `<query>` and an identical copy of the sequence. The task uses label masking, where the loss is computed only on the N answer tokens. Models are pretrained with a context length of 1024, a total batch size of 32, AdamW optimizer ($\beta = 0.9, 0.98$ and $\varepsilon = 10^{-6}$), weight decay of 0.03, learning rate warmup for the first 1000 steps, and cosine decay to 10%. Training duration is set to 50k steps, and the best results are reported across learning rates $\{0.0005, 0.001, 0.002, 0.005\}$.

For this task, we also assessed the models' ability to correctly copy the first $t = 1, 2, 4$ tokens within the sequence. As shown in Figure 19, these results are nearly identical to those in Figure 5.



(a) Evaluated with $t = 1$ (b) Evaluated with $t = 2$ (c) Evaluated with $t = 4$

Figure 19: A trivial experiment for copying 500 tokens, evaluated only on correctly copying the first t tokens.

Task 1-hop-L and 2-hop-L. In the real-life experiment (Section 9), we evaluated models' performance on extremely simple 1-hop and 2-hop information retrieval tasks.

For the 1-HOP-L task, we prepared five random birth year statements of the form "[name] was born in the year of [year]," where names are generated as random combinations of first, middle, and last names, and years are sampled uniformly from 1950 to 2009. The five sentences were embedded into random Wikipedia documents of length L tokens, with each statement inserted between sentences at up to five randomly chosen positions. Finally, the model was prompted with "`\n\n`Answer me: name was born in the year of" to test its ability to retrieve the birth year. This setup closely replicates the needle-in-a-haystack task [28], but we intentionally made the task more "natural English" by using birth years (commonly found in pretraining datasets like Wikipedia) instead of abstract multi-digit numbers.

For the 2-HOP-L task, three random birth year statements were prepared in the same format as above. This was followed by three equivalence statements of the form "[name1] was born in the same year as [name2]," where random names were generated such that the equivalences formed a bijection between the two sets of three random names. To simplify the task, we did not shuffle the ordering of the statements; the three equivalence statements always followed the three original ones. These six sentences were then embedded into random Wikipedia documents of length L tokens,

[28] This is crucial to ensure that the model is generating the genuine probabilistic distribution of sentences; if using temperature 0 for instance, the generation is always a fixed string, and accuracy would be either 0 or 100% forever.

inserted at up to six different positions between sentences, respecting their original order. At the end, the model was prompted with "`\n\n`Answer me: name was born in the year of" to test its ability to infer and retrieve the correct birth year. To further assist the model, an instructional statement was added at the beginning of the context. [29] This design represents arguably the simplest possible and most natural 2-hop in-context reasoning task, yet even with $L = 0$, models largely failed to perform, as demonstrated in Figure 17.

SlimPajama and FineWeb-edu 100B. The SlimPajama dataset is sourced directly from Huggingface (`cerebras/SlimPajama-627B`), with the first 100M data samples selected. This subset provides more than 100B training tokens, sufficient for our pretraining experiments. Similarly, the FineWeb-Edu dataset is obtained from Huggingface (`HuggingFaceFW/fineweb-edu`), utilizing its predefined 100B split, which contains more than 100B tokens for pretraining.

Following a standard pretraining protocol, the data is processed sequentially by tokenizing it in the provided order. It is then concatenated into a single continuous text stream, from which random subsequences of length 4096 are extracted to serve as training windows. These sampled windows are used for pretraining across all architectures.

For each pretraining run, we use a total batch size of 48 and the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-6}$. A weight decay of 0.03 is applied. Training is conducted with two learning rates $\{0.001, 0.002\}$ for Llama and GPT models, and with *three* learning rates $\{0.0005, 0.001, 0.002\}$ for linear models (Mamba, GLA, GatedDeltaNet) to provide a stronger comparison. For each evaluation task, we report the best accuracy achieved across the two or three runs. Each model is trained for 510,000 steps, yielding a total of $4096 \times 48 \times 510{,}000 = 100.2$B tokens processed per run.

To ensure fair comparisons across architectures, we fix the random seed during pretraining. This guarantees that, under the same seed, all architectures are trained on identical data and in identical order, even in cases where pretraining jobs are interrupted and resumed. This setup minimizes potential discrepancies due to differences in training data and maximally ensures consistency.

For the Llama(RoPE) experiments, we additionally test across eight random seeds to evaluate variability in performance. These results are presented in Figure 17.

SlimPajama 300B. We briefly tested a pretraining setup using 300B tokens from SlimPajama with a 1.7B-parameter Llama model (24 layers, 24 heads, 2304 dimensions), both with and without Canon layers. Training utilized three learning rates $\{0.0005, 0.001, 0.002\}$, a batch size of 128, 575,000 steps, and a gradient accumulation factor of 5. The results were similarly noisy to the 100B pretraining case (e.g., models failed at 2-hop reasoning), so we excluded them from Figure 17 for clarity. However, we include their Babilong results in Figure 25. Unfortunately, this remains the largest experiment we are able to support at the moment.

C Details on Architectures Used

Llama/GPT Models. In this paper, "Llama(RoPE)" refers to the Huggingface implementation LlamaForCausalLM, which employs rotary embeddings across all hidden dimensions and utilizes gated MLP layers. We did not enable group-query attention, as this study focuses on smaller-scale models. The intermediate size is set to $\frac{8d}{3}$, ensuring that each MLP layer contains $8d^2$ trainable parameters, consistent with standard MLP layers. "Llama(NoPE)" refers to the same architecture with rotary embedding completely disabled. "Llama(RoPE)♩" refers to the version where rotary

[29] "You will be asked questions about people's birth years, and the birth year descriptions are hidden in some random text. Some people's birth years are directly given, while others are given in the form that 'name1' was born in the same year as 'name2'. "

embeddings are applied to only a quarter of the dimensions. The variants ♩, ♩♩, and ♩♩♩represent differing fractions of dimensionality on which RoPE is enabled, as described in the main paper.

For direct comparisons, "GPT2(RoPE)" refers to the Llama architecture with gated MLP layers replaced by standard MLP layers. The intermediate size in these models is set to $4d$, ensuring that each MLP layer contains $8d^2$ trainable parameters.[30]

We denote "GPT2(RoPE,R2)" as the GPT2(RoPE) model with its `silu` activation replaced by ReLU^2, following the design proposed in Primer [57]. Similarly, "Llama(RoPE,R2)" refers to Llama(RoPE) with ReLU^2 in place of `silu`.

ALiBi and H-ALiBi. For ALiBi [44], we follow the original recommendation of using a geometric sequence $2^{-8/n}$ for an n-head Transformer, which determines how each head is biased toward local context. For H-Alibi [30], we use their proposed strategy of restricting the h-th head to attend only to the nearest h tokens, and applied to half of the heads. (We briefly tested applying this to one-third of the heads instead, but observed slightly worse performance.)

Mamba Models. For "Mamba2," we use the Huggingface implementation Mamba2ForCausalLM, with recommended configuration parameters:

`ssm_state_size=64`, `num_heads=16`, and `head_dim=hidden_size * 2 / num_heads`.

This setup ensures each Mamba layer has $6d^2 + o(d^2)$ trainable parameters. We briefly tested `num_heads=8` but observed worse results, and also tested "Mamba1," which consistently underperformed compared to Mamba2. For these reasons, neither configuration is included in this paper. The model initialization range follows the default settings provided in the implementation.

For "Mamba2(mlp)," we use the same Huggingface implementation but alternate between Mamba SSM layers and gated MLP layers. The intermediate size for the MLP layers is set to $2d$, ensuring each MLP layer contains $6d^2$ trainable parameters. This ensures that ℓ-layer d-dimensional Llama(RoPE) and Mamba2(mlp), as well as 2ℓ-layer d-dimensional Mamba2, have comparable parameter counts.

Mamba1. We briefly tested Mamba1 and found it consistently outperformed by Mamba2 in our pretraining playground, so we excluded it from main results. Notably, removing its `conv1d` layer also degrades Mamba1 to GLA-level performance.

Mimetic initialization. Following [63], we enabled $A \approx 1$ (via c=8), $\Delta \approx 1$ (via b_Δ=0.54), $W_C^\top W_B \approx I$, and conv1d $\approx I$. We also tested c=4 and c=2 but observed no improvement.

GLA Models. For Gated Linear Attention (GLA), we use the official GitHub implementation.[31] The default configuration uses 4 linear attention heads, as suggested in their implementation. We briefly tested configurations with 8 attention heads but found that these consistently degraded performance. The default GLA implementation has `conv1d` disabled (the functionality was not part of the original publication [68]), although their codebase supports `conv1d`, which we explicitly tested in this paper.

For GLA(elu) experiments in the ablation studies, we replaced the default feature map with $elu(x) + 1$, and conducted evaluations with and without `conv1d` and Canon layers.

Unless otherwise stated (e.g., in Task Capo), we do not tie weights between the embedding and output layers in any of the architectures (e.g., Llama, Mamba, GLA). Additionally, no tokenizers are used during pretraining except for Task Capo.

[30] The original GPT2 architecture differs from Llama in other minor ways, such as using GeLU activation and slightly different initialization. We do not investigate these small architectural differences in this paper.

[31] See `https://github.com/fla-org/flash-linear-attention`. From March to May 2025, the repo authors updated the initializer range to 0.006 (from the previously popular 0.02), which we found negatively impacted performance for our tested model sizes. For our experiments, we reverted to 0.02. The repo authors also restored this setting to 0.02 on May 3, 2025.

Task Capo. The knowledge capacity task involves pretraining fake biographies, following the method described in [7]. For consistency, GPT2Tokenizer is used throughout, along with weight tying between embedding and output layers.[32]

Since CAPO evaluates bit-per-parameter knowledge capacity, we scale both model and data sizes *more aggressively* to assess scaling behavior. Following [7], we use the notation ℓ-h to denote Llama with ℓ layers, hidden size $64h$, and h heads, extending this convention to GLA and Mamba2.[33]

For GPT2 experiments in Figure 11, we use the original GPT2 architecture with added RoPE, consistent with [7]. Mixture-of-Experts (MoE) experiments use the `tutel` [29] package with 32 standard MLP experts, configured with $topk = 1$ and $cap\_factor = 2$.

Real-Life Experiments. For pretraining experiments on SlimPajama and FineWeb-Edu, we use all the architectures listed above alongside the Llama2 tokenizer (with vocab size 32,000) [62]. Weight tying is disabled to maintain consistency with prior works (e.g., [9, 69] and references therein). We include GatedDeltaNet [69] in the real-life experiments, using its official GitHub implementation with head dimensions set to $= \frac{3d}{4 \cdot \text{num\_heads}}$, ensuring that each block layer contains $12d^2 + o(d^2)$ trainable parameters for direct comparison with other architectures.

The architectural configurations used in the real-life experiments (approximately 1.35B parameters each) are summarized below:

- **Llama(RoPE/NoPE):** 24 layers, 32 heads, 64 dimensions per head (total hidden size: 2048).
- **GLA:** 24 layers, 4 heads, total hidden size: 2048.
- **Mamba2:** 48 layers, 16 heads, total hidden size: 2048.
- **Mamba2(mlp):** 24 layers, 16 heads, total hidden size: 2048.
- **GatedDeltaNet:** 24 layers, 12 heads, total hidden size: 2048.

Canon Implementations. Canon layers in this paper are implemented using PyTorch's `nn.Conv1D` with kernel size 4, zero padding, and default initialization. A good source of its CUDA fast implementation is available in the `fla-org` repository.[34] Canon layers are applied after layer normalization (if present, e.g., Canon-A/C) and before any non-linearity (if present, e.g., Canon-B/D).

We refer to `cst-Canon` as the constant, untrained version of Canon(res), where the convolution weights are fixed to PyTorch's default values. We plan to open-source our synthetic playground and evaluation toolkit after this paper's release, following code cleanup and documentation.

[32] Weight tying minimizes capacity loss in smaller models, although this effect is minor.

[33] In GLA, ℓ-h refers to ℓ layers with hidden size $64h$ and 4 fixed attention heads, while in Mamba2, ℓ-h corresponds to 2ℓ layers with hidden size $64h$. Thus, the ℓ-h convention ensures model sizes remain comparable across architectures for controlled evaluations.

[34] `https://github.com/fla-org/flash-linear-attention/blob/main/fla/modules/convolution.py`

D Missing Experiments

D.1 Intentionally Omitted Experiments

In this sub-section, we present missing figures that were intentionally omitted from the main body of the paper for the sake of clarity and conciseness.

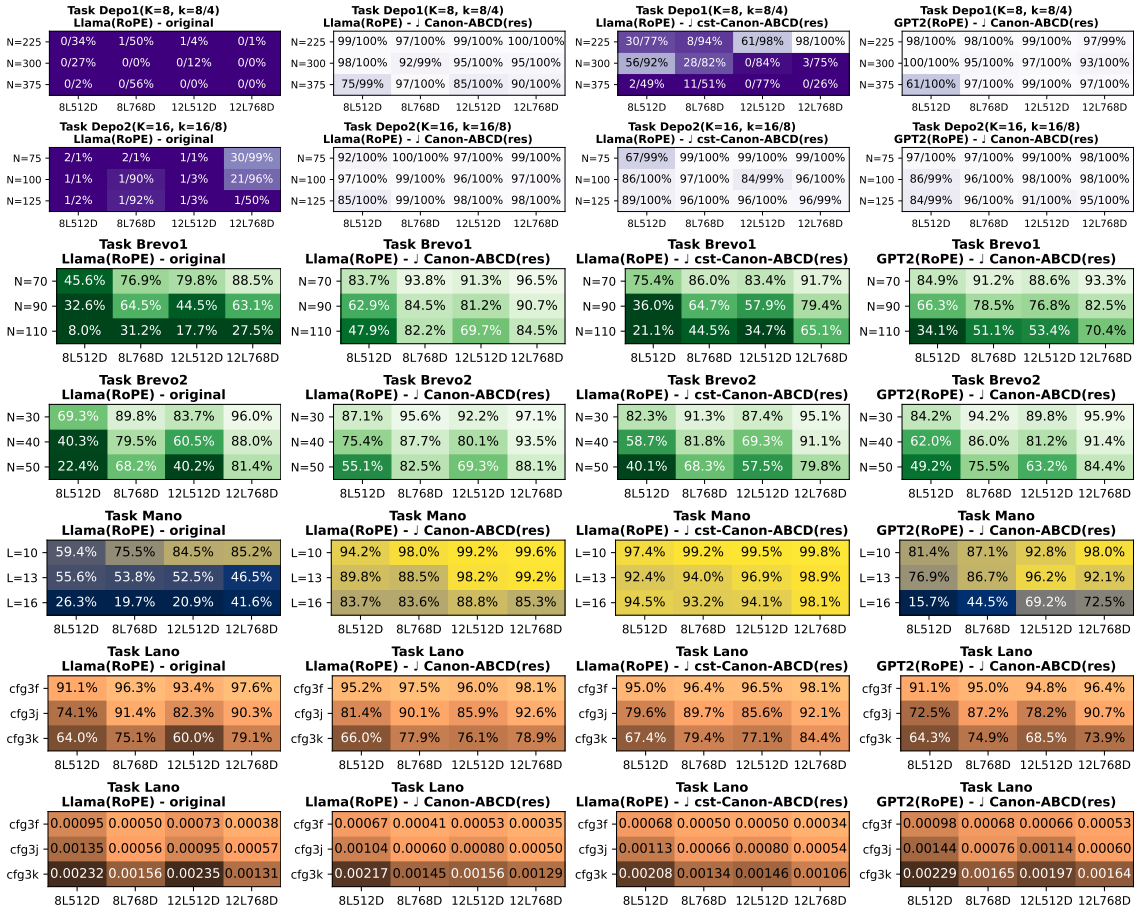


Figure 20: **Columns 1,2,3:** Constant Canon implementation (random, *non-trained* average of the past 3 tokens, denoted **cst-Canon**) already achieves strong performance, clearly outperforming vanilla Llama.
Columns 1,2,4: Canon layers also perform strongly on GPT2 models (with standard MLP). Our playground reveals standard MLP is slightly weaker than gated MLP, especially in knowledge manipulation (cf. Result 5).

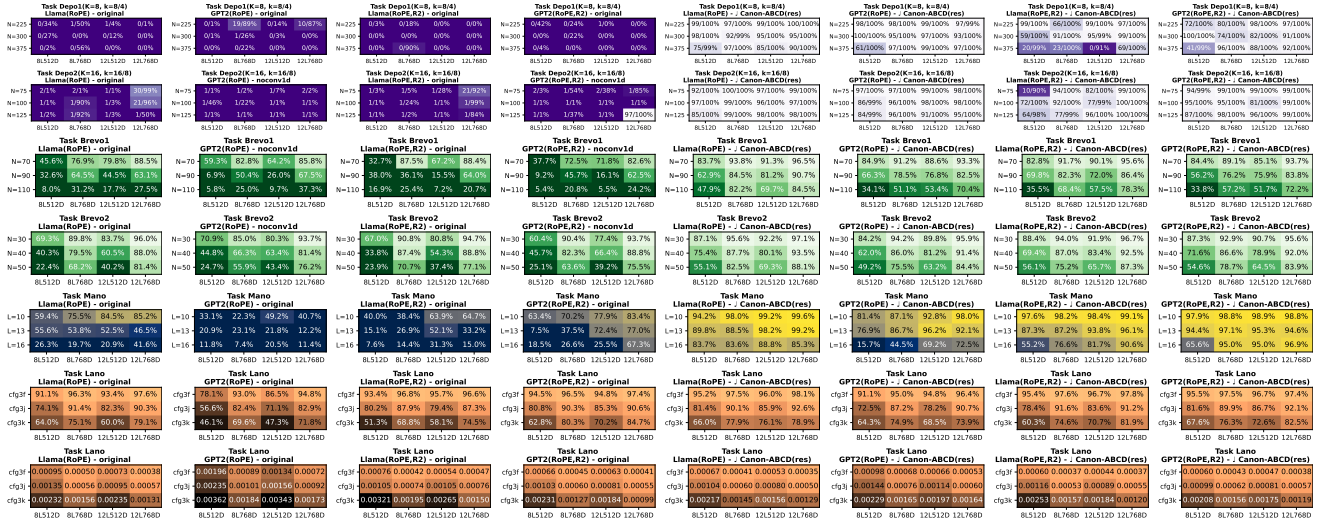


Figure 21: Effect of ReLU^2 activation on standard vs. gated MLP. **Columns 1→2, 5→6**: gated MLP outperforms standard MLP with `silu`. **Columns 2→4, 6→8**: adding ReLU^2 to standard MLP yields slight gains. **Columns 1→3, 5→7**: adding ReLU^2 to gated MLP hurts performance.

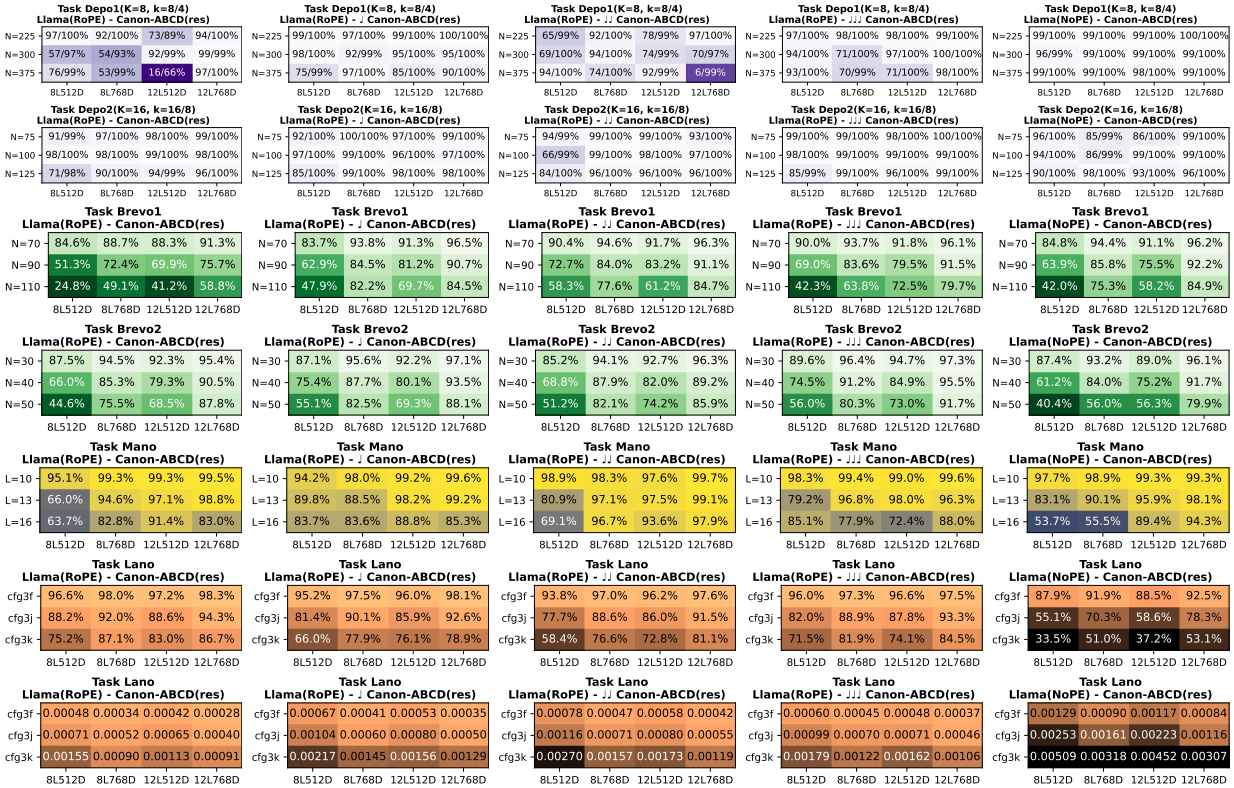


Figure 22: Transformer+Canon with varying RoPE configurations. From left to right: (1) RoPE; (2) RoPE♩: half of heads each with half RoPE dimensions; (3) RoPE♫: a quarter of heads with full RoPE dimensions; (4) RoPE♬: all heads each with quarter RoPE dimensions; (5) NoPE.

Conclusion: Canon layers eliminate the need for extensive RoPE usage, and reducing RoPE usage to 1/4 is even preferable, outperforming both full RoPE and NoPE setups. Among these reduced RoPE variants, RoPE♩ achieves slightly better overall performance.

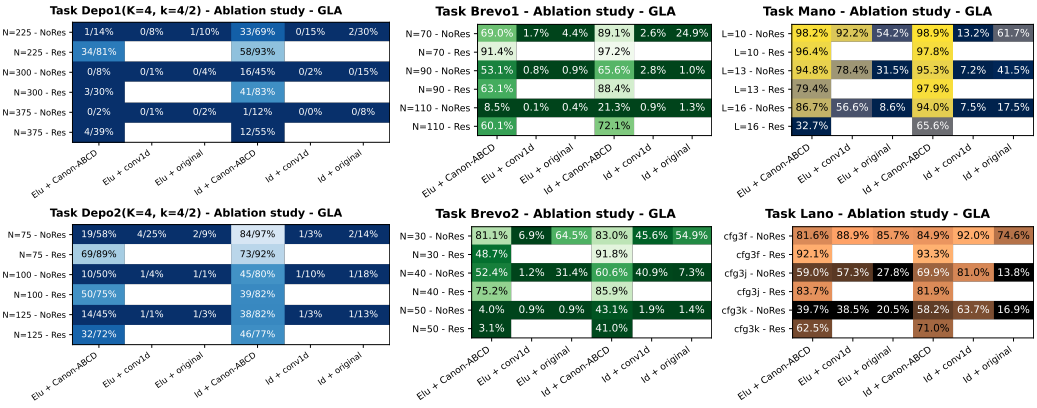


Figure 23: Ablation study on 12-layer, 768-dimensional GLA with Canon/conv1d layers, residual vs. non-residual, identity feature map vs non-linear ($\phi(x) = \text{elu}(x) + 1$) feature map.

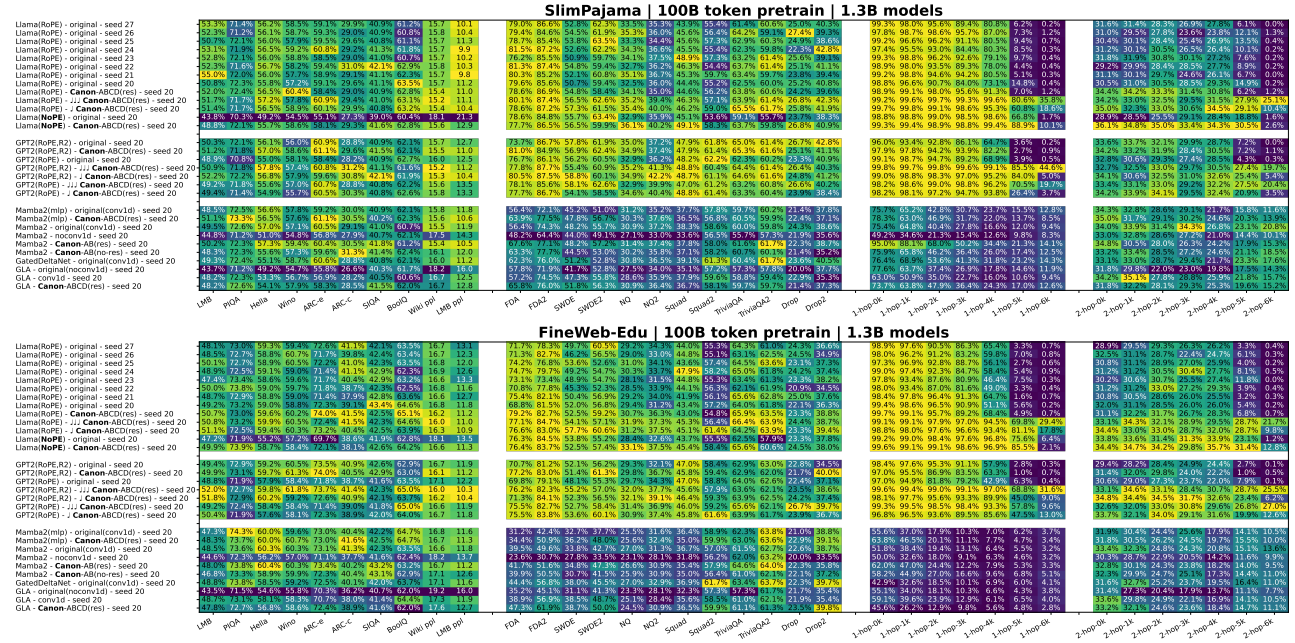


Figure 24: This is identical to Figure 17 but additionally includes GPT2(RoPE) models—identical to Llama(RoPE) but using standard MLPs—and GPT2(RoPE,R2), which replaces `silu` with ReLU^2 activation. **Key conclusions remain unchanged**: reducing RoPE improves length generalization, and many architectural differences (e.g., standard vs. gated MLP, `SiLU` vs. ReLU^2) are buried in noise.

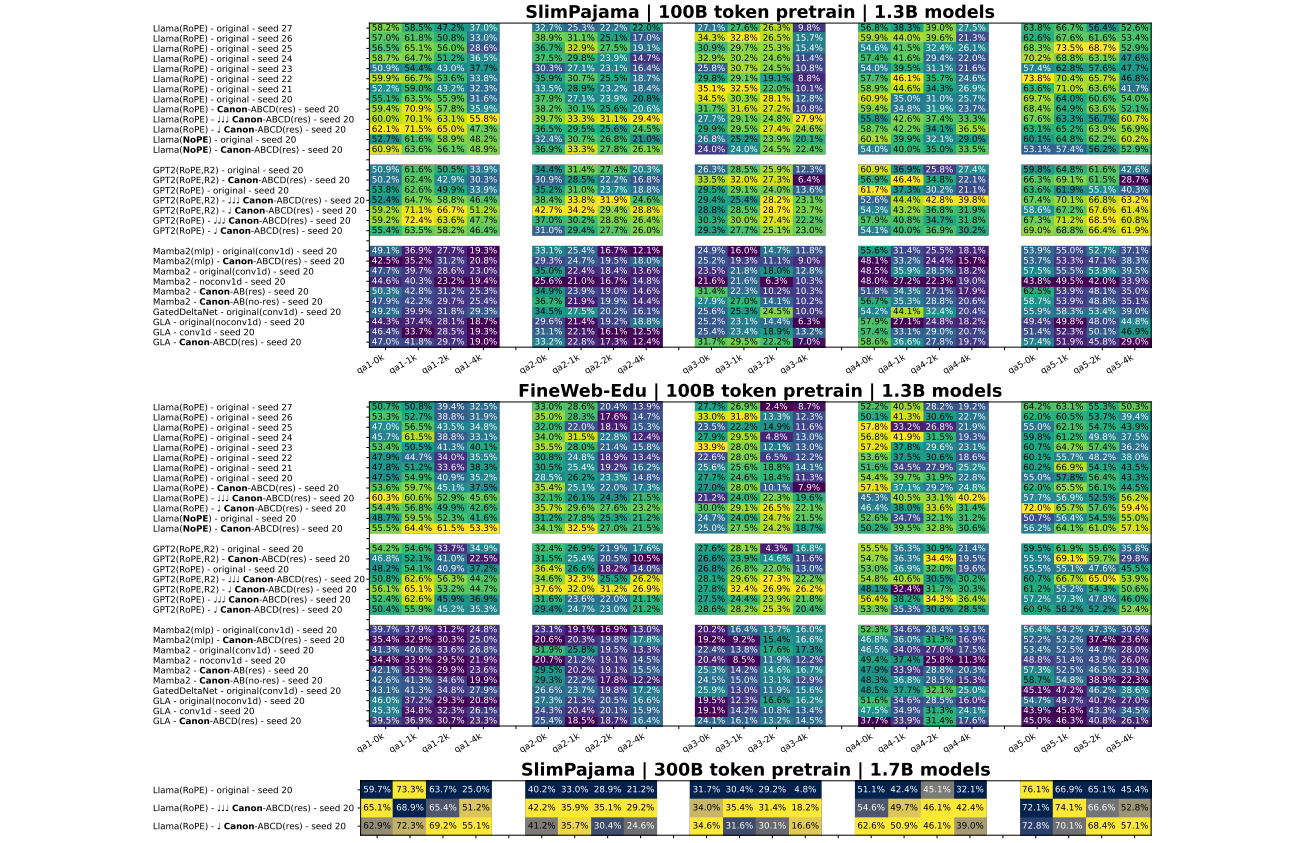


Figure 25: Results on the Babilong dataset evaluating multi-hop reasoning across varied junk context lengths. Most architectural comparisons are statistically insignificant. Key findings include:

1. Linear models consistently underperform Transformers, even in short contexts without junk.
2. Models with reduced RoPE (NoPE, RoPE$\blacklozenge$) achieve notable improvements in long-context accuracy.

D.2 Even More Ablation Studies

In this section, we present additional pretraining experiments, including full-scale ablation studies, KL-divergence evaluations for MANO tasks, and other analyses. These results were omitted from the main body of the paper to maintain focus and clarity; however, they may be of interest to certain readers (or at least to ourselves!) who seek deeper insights into the experimental setup and findings. For the sake of completeness, these supplementary results are provided here.

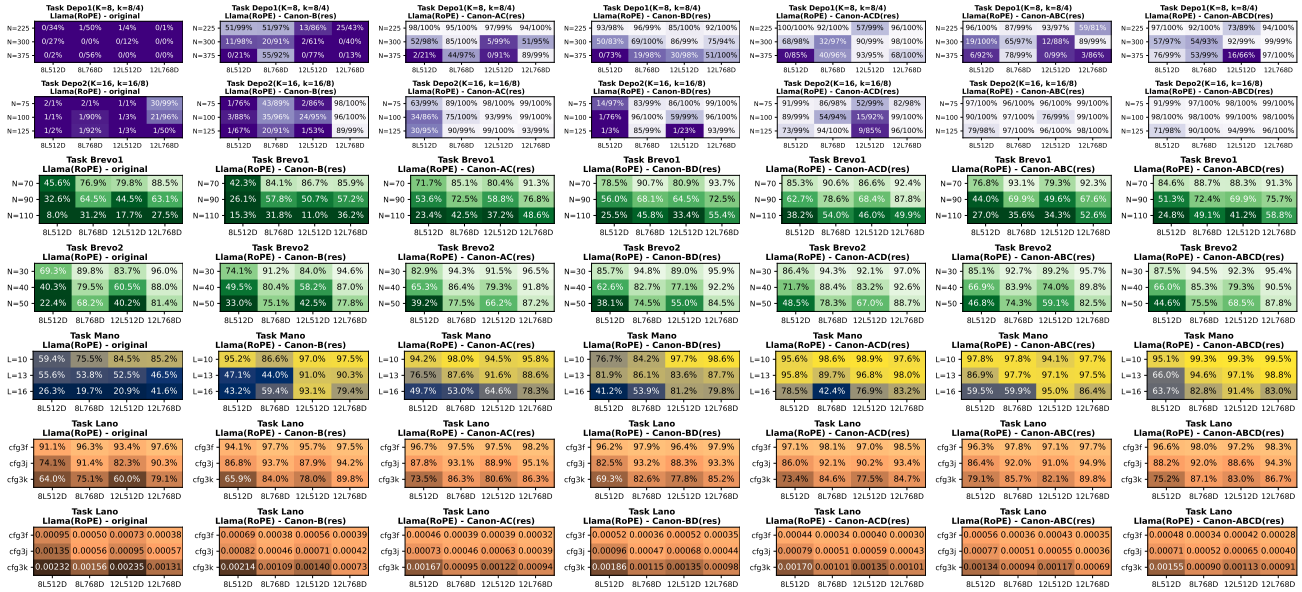


Figure 26: Llama(**RoPE**) family: (from left to right) original, Canon-B, -AC, -BD, -ACD, -ABC, -ABCD. This figure complements Figure 10 and gives more technical details.

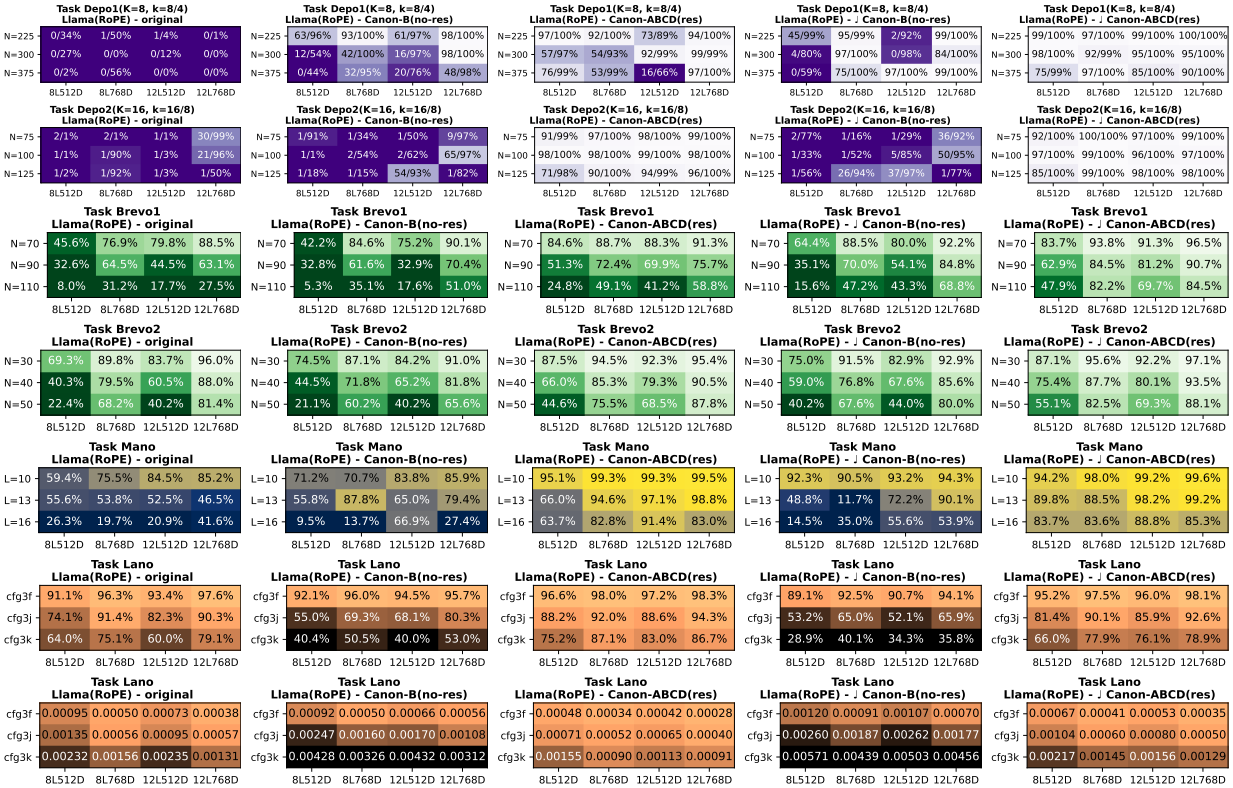


Figure 27: Llama(**RoPE**) family: (left to right) original, Canon-B(no-res), Canon-ABCD(res), ♩Canon-B(no-res), ♩Canon-ABCD(res).
This figure complements Figure 10 and directly compares to Primer [57] (i.e., Canon-B(no-res)), showing its inefficiency and highlighting: (1) Canon layers are *not tied* to Attention; (2) Canon(res) at multiple points is safe and more effective.

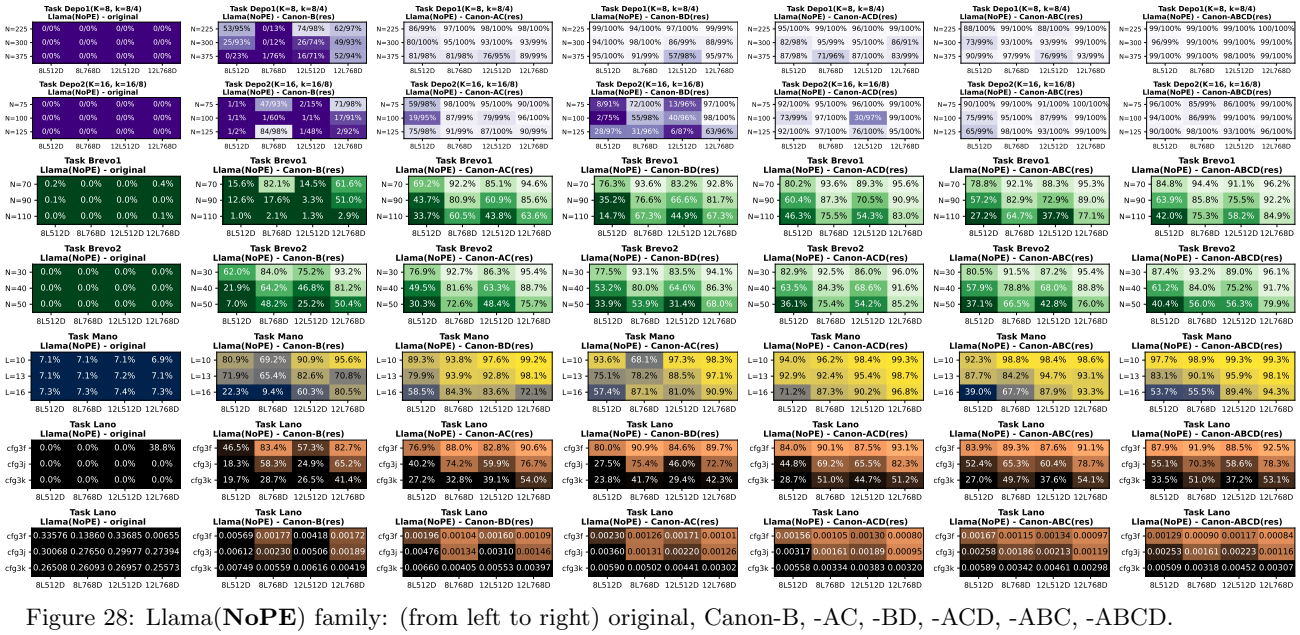


Figure 28: Llama(**NoPE**) family: (from left to right) original, Canon-B, -AC, -BD, -ACD, -ABC, -ABCD.
This figure complements Figure 10 and gives more technical details.

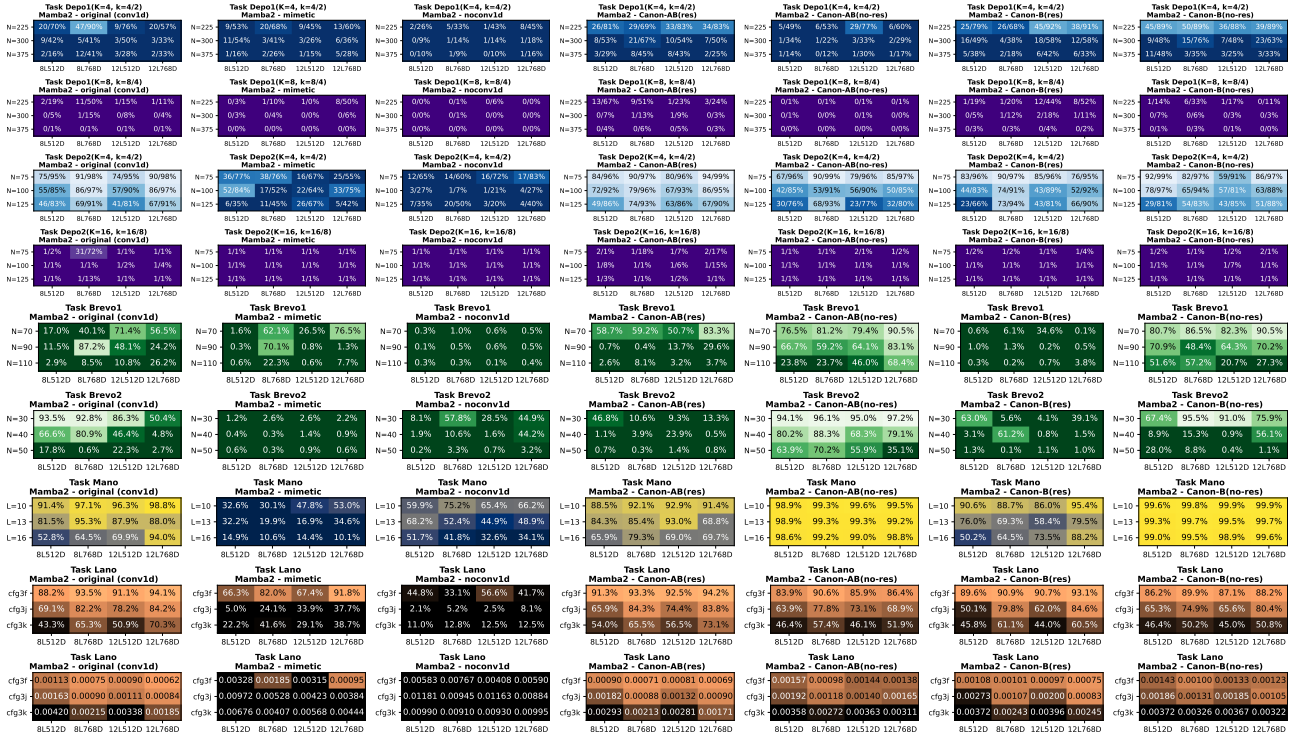


Figure 29: **Mamba2 variants** (left to right): original (conv1d), original (conv1d) + mimetic, no conv1d, Canon-AB(res), Canon-AB(no-res), Canon-B(res), Canon-B(no-res).

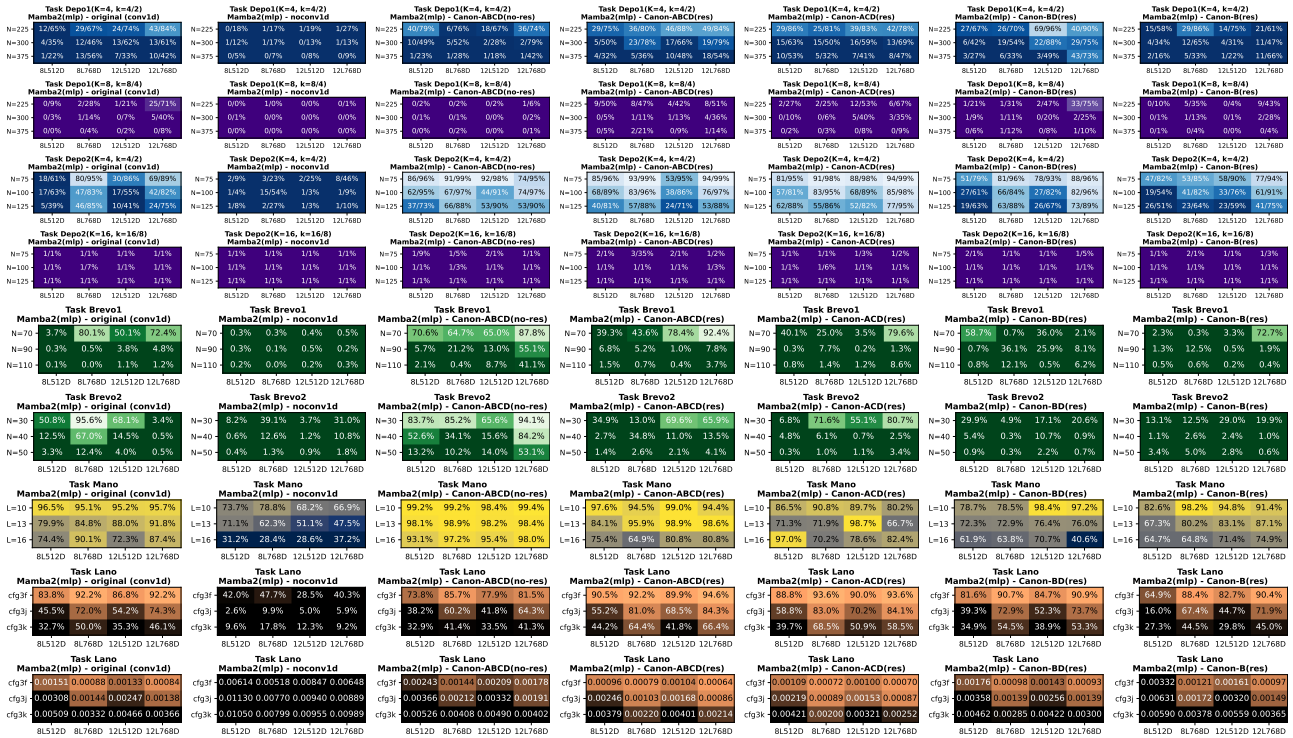


Figure 30: **Mamba2(mlp) variants** (left to right): original (conv1d), no conv1d, Canon-ABCD(no-res), Canon-ABCD(res), Canon-ACD(res), Canon-BD(res), Canon-B(res).

References

[1] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. Can SGD Learn Recurrent Neural Networks with Provable Generalization? In *NeurIPS*, 2019. Full version available at `http://arxiv.org/abs/1902.01028`.

[3] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of Language Models: Part 1, Learning Hierarchical Language Structures. *ArXiv e-prints*, abs/2305.13673, May 2023. Full version available at `http://arxiv.org/abs/2305.13673`.

[4] Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. In *COLT*, 2023. Full version available at `http://arxiv.org/abs/2001.04413`.

[5] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of Language Models: Part 3.1, Knowledge Storage and Extraction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML 2024, 2024. Full version available at `http://arxiv.org/abs/2309.14316`.

[6] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of Language Models: Part 3.2, Knowledge Manipulation. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. Full version available at `http://arxiv.org/abs/2309.14402`.

[7] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of Language Models: Part 3.3, Knowledge Capacity Scaling Laws. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. Full version available at `http://arxiv.org/abs/2404.05405`.

[8] Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*, 2024.

[9] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[11] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[12] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022. URL `https://arxiv.org/abs/2204.06745`.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[15] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[16] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2924–2936, 2019. doi: 10.18653/v1/N19-1300.

[17] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and

Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[19] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. URL https://arxiv.org/abs/2405.21060.

[20] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[21] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

[22] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[23] Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022. URL https://arxiv.org/abs/2212.14052.

[24] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

[25] Olga Golovneva, Tianlu Wang, Jason Weston, and Sainbayar Sukhbaatar. Multi-token attention. *arXiv preprint arXiv:2504.00927*, 2025.

[26] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. URL https://arxiv.org/abs/2312.00752.

[27] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*, 2020.

[28] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[29] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. *CoRR*, abs/2206.03382, June 2022. URL https://arxiv.org/pdf/2206.03382.pdf.

[30] Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

[31] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023. doi: 10.1145/3571730. URL https://doi.org/10.1145/3571730.

[32] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[33] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of*

the Association for Computational Linguistics (Volume 1: Long Papers), pages 1601–1611, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

[34] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[35] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[36] Yury Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.

[37] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

[38] Nayoung Lee, Ziyang Cai, Avi Schwarzschild, Kangwook Lee, and Dimitris Papailiopoulos. Self-improving transformers overcome easy-to-hard and length generalization challenges. *arXiv preprint arXiv:2502.01612*, 2025. URL https://arxiv.org/abs/2502.01612.

[39] OpenAI. Gpt-4 technical report, 2023.

[40] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, 2016. doi: 10.18653/v1/P16-1144.

[41] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

[42] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[43] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022. URL https://arxiv.org/abs/2201.02177.

[44] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[45] Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically gated recurrent neural network for sequence modeling. *Advances in Neural Information Processing Systems*, 36:33202–33221, 2023.

[46] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[47] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124/.

[48] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

[49] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial

winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

[50] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, 2019. doi: 10.18653/v1/D19-1454.

[51] Eshika Saxena, Alberto Alfarano, Emily Wenger, and Kristin Lauter. Teaching transformers modular arithmetic at scale. *arXiv preprint arXiv:2410.03569*, 2024. URL `https://arxiv.org/abs/2410.03569`.

[52] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[53] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[54] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[55] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2016.

[56] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

[57] DR So, W Manke, H Liu, Z Dai, N Shazeer, and QV Le. Primer: Searching for efficient transformers for language modeling. arxiv 2021. *arXiv preprint arXiv:2109.08668*.

[58] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

[59] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

[60] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

[61] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[62] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[63] Asher Trockman, Hrayr Harutyunyan, J Zico Kolter, Sanjiv Kumar, and Srinadh Bhojanapalli. Mimetic initialization helps state space models learn to recall. *arXiv preprint arXiv:2410.11135*, 2024.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[65] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[66] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[67] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22–31, 2021.

[68] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

[69] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

[70] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

[71] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of Language Models: Part 2.1, Grade-School Math and the Hidden Reasoning Process. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. Full version available at `https://arxiv.org/abs/2407.20311`.

[72] Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of Language Models: Part 2.2, How to Learn From Mistakes on Grade-School Math Problems. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR 2025, 2025. Full version available at `http://arxiv.org/abs/2408.16293`.

[73] Ping Yu, Jing Xu, Jason Weston, and Ilia Kulikov. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024.

[74] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

[75] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019. doi: 10.18653/v1/P19-1472.

[76] Yu Zhang, Songlin Yang, Rui-Jie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *Advances in Neural Information Processing Systems*, 37:116870–116898, 2024.

[77] Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu^2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024.

[78] Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly. *arXiv preprint arXiv:2402.09371*, 2024.