

Olmo Hybrid: From Theory to Practice and Back

William Merrill♥¹ Yanhong Li♥¹ Tyler Romero♥¹ Anej Svete♥^{1,4} Caia Costello♥³
Pradeep Dasigi¹ Dirk Groeneveld¹ David Heineman¹ Bailey Kuehl¹ Nathan Lambert¹
Chuan Li³ Kyle Lo^{1,2} Saumya Malik¹ DJ Matusz³ Benjamin Minixhofer^{1,5}
Jacob Morrison^{1,2} Luca Soldaini¹ Finbarr Timbers¹ Pete Walsh¹ Noah A. Smith^{1,2}
Hannaneh Hajishirzi^{1,2} Ashish Sabharwal♥¹

¹Allen Institute for AI ²University of Washington ³Lambda ML Team
⁴ETH Zürich ⁵University of Cambridge

✉ willm@allenai.org ♥ marks core contributors. See author contributions [here](#).

🗂️ **Collection:** [Olmo-Hybrid-7B](#)

📊 **Training Logs:** [Olmo-3-7B-vs-Olmo-Hybrid](#)

🔗 **Training Code:** [OLMo-core](#) (pretrain, mid-train, long context, sft-think, sft-instruct)

Abstract



Recent work has demonstrated the potential of non-transformer language models, especially linear recurrent neural networks (RNNs) and hybrid models that mix recurrence and attention. Yet there is no consensus on whether the potential benefits of these new architectures justify the risk and effort of scaling them up. To address this, we provide evidence for the advantages of hybrid models over pure transformers on several fronts. First, theoretically, we show that hybrid models do not merely inherit the expressivity of transformers and linear RNNs, but can express tasks *beyond* both, such as code execution. Putting this theory to practice, we train **Olmo Hybrid**, a 7B-parameter model largely comparable to Olmo 3 7B but with the sliding window layers replaced by Gated DeltaNet layers. We show that Olmo Hybrid outperforms Olmo 3 across standard pretraining and mid-training evaluations, demonstrating the benefit of hybrid models in a controlled, large-scale setting. We find that the hybrid model scales significantly more efficiently than the transformer, explaining its higher performance. However, it's unclear why greater expressivity on specific formal problems should result in better scaling or superior performance on downstream tasks unrelated to those problems. To explain this apparent gap, we return to theory and argue why increased expressivity should translate to better scaling efficiency, completing the loop. Overall, our results suggest that hybrid models mixing attention and recurrent layers are a powerful extension to the language modeling paradigm: not merely to reduce memory during inference, but as a fundamental way to obtain more expressive models that scale better during pretraining.

Contents

1	Introduction	3
2	Olmo Hybrid Overview	4
2.1	Architecture	4
2.2	Training Overview	4
2.3	Olmo Hybrid Results	5
3	Expressive Power of Hybrid Models	7
3.1	Background: Expressive Power of Transformers	8
3.2	Background: Expressive Power of Linear RNNs	9
3.3	Hybrid Models are More Than the Sum of Their Parts	10
3.4	Expressive Power of Padded Hybrid Models	11
3.5	Synthetic Evaluations	13
3.6	Discussion: Pushing the Expressivity-Parallelism Frontier	13
4	Scaling Behavior of Hybrid Models	14
4.1	Scaling Laws for Hybrid Models	14
4.2	Theory: Increased Expressive Power Improves Scaling	18
4.3	Discussion: From Expressivity to Scaling	22
5	Other Research Questions	22
5.1	RNN and Hybridization Architecture Choices	22
5.2	Olmo Hybrid vs. Other Open Models	25
5.3	Post-Training Olmo Hybrid	26
6	Conclusion	29
A	Training Olmo Hybrid	39
A.1	Pretraining	39
A.2	Mid-Training and Long Context Extension	40
A.3	Post-Training	41
A.4	Evaluation Details	41
B	Proofs: Expressive Power of Hybrid Models	41
B.1	Limitations of Transformers and RNNs	42
B.2	Power of Hybrid Models	44
B.3	Padded Hybrid Models	45
C	Additional Details: Synthetic Evaluations	46
C.1	Tasks and Evaluation Protocol	46
C.2	Model Architectures	46
C.3	Training Details	46
D	Scaling and Ablation Experiments: Details and Additional Results	49
D.1	Additional Results	49
D.2	Details on the Scaling Law Fits	50
D.3	Architectural Details	53
D.4	Parameter Count and FLOP Computations	55
E	Proofs: Increased Expressive Power Improves Scaling	61
E.1	Scaling Law with Tasks Learned	62
E.2	Parameter Scaling Law	64
E.3	Data Scaling Law	64
E.4	Main Results	67

1 Introduction

Transformers have become the backbone architecture for language models (Radford and Narasimhan, 2018), after initially eclipsing recurrent neural networks in machine translation (Vaswani et al., 2017). Recently, theoretical and empirical work has characterized the limitations of transformers (Strobl et al., 2024) and the potential for alternative recurrent architectures to overcome them (Merrill et al., 2024; Grazzi et al., 2025). Transformers lack the expressive power to robustly represent state tracking tasks, which require sequential computation (Merrill and Sabharwal, 2023; Chiang, 2025). While the parallelism of transformers is important for efficient training, a longstanding issue at inference time is their quadratic scaling with context length. Modern RNNs strike a better balance: their compute scales *linearly* with context length, and, unlike classical RNNs (Elman, 1990; Hochreiter and Schmidhuber, 1997), they can be parallelized through linear gating (Bradbury et al., 2017; Katharopoulos et al., 2020; Gu et al., 2022; Gu and Dao, 2024). Moreover, these linear RNNs can also express state tracking tasks, giving an expressivity advantage over transformers (Merrill et al., 2024; Grazzi et al., 2025; Peng et al., 2025).

However, switching from transformers to RNNs is not a free lunch, as RNNs (even linear ones) struggle with copying and recall tasks due to their bounded state (Arora et al., 2024a; Jelassi et al., 2024). This has led to the exploration of *hybrid models* that mix attention and linear RNN layers in order to leverage the benefits of both architectural primitives. Recently, hybrid models have been trained at scales up to 9B active parameters and 36T tokens (e.g., Samba, Ren et al., 2025; Nemotron-H, NVIDIA, 2025; Qwen3-Next, Qwen Team, 2025; Kimi Linear, Kimi Team, 2025; and Qwen 3.5, Qwen Team, 2026) with encouraging results. Yet there is not yet consensus on whether the advantages of hybrid models justify the cost and risk of switching to a fundamentally different architecture, in part because a controlled comparison between transformer and hybrid LMs at large scale is lacking.

To rectify this gap, we introduce **Olmo Hybrid**, a family of model artifacts comparable to Olmo 3 except that the architecture interleaves Gated DeltaNet (GDN, Yang et al., 2025a; Grazzi et al., 2025) layers with attention layers at a 3:1 ratio in place of the sliding-window-attention layers used in Olmo 3. We train Olmo Hybrid 7B on up to 6T tokens, finding large improvements in token efficiency (and thus also compute efficiency): Olmo Hybrid matches Olmo 3 7B on MMLU (Hendrycks et al., 2021) with 49% fewer training tokens. After pretraining, this token efficiency translates to improvements on MMLU and other evaluations, with gains persisting after mid-training; the final Olmo Hybrid base checkpoint outperforms Olmo 3 across all aggregated domains of OLMOBASEEVAL. Beyond these standard base model evaluations, Olmo Hybrid shows large gains in long-context ability, with a 14.1% improvement on RULER 64k over Olmo 3. Because Olmo Hybrid is closely comparable to Olmo 3 apart from its hybrid architecture, we take these results as strong evidence in favor of hybrid models over pure transformers.

Beyond large-scale experiments, we also present theoretical results and fully controlled scaling studies to explain these performance gains. Past theoretical work has shown attention and recurrence have complementary strengths (Merrill et al., 2024; Grazzi et al., 2025). Mixing them is, thus, a natural way to reap the benefits of both primitives. We extend this with novel theory showing that hybrid models are more powerful than the sum of their parts: there are formal problems related to code evaluation that neither transformers nor GDN can express on their own, but which hybrid models can represent theoretically and learn empirically. This greater expressivity does not immediately imply that hybrid models should be better LMs. We therefore run controlled scaling studies comparing hybrid models to transformers and show that they indeed attain better token efficiency, consistent with our observations from the Olmo Hybrid pretraining run. It is not clear *prima facie* why greater expressivity on specific formal problems should improve scaling efficiency on benchmarks like MMLU. To fill this gap, we develop a formal explanation of how increasing an architecture’s expressivity can improve data and compute efficiency for loss and downstream tasks—even on tasks unrelated to new problems expressible by the model.

Taken together, our results suggest that hybrid models dominate transformers both theoretically, in their balance of expressivity and parallelism, and empirically, in terms of benchmark performance and long-context abilities. We believe these findings position hybrid models for wider adoption and call on the research community to pursue further architecture research.

2 Olmo Hybrid Overview

2.1 Architecture

Our architecture matches that of Olmo 3 7B (see Olmo Team, 2025 for details), except that 75% of layers use GDN heads in place of attention heads. The layers alternate so that 3 GDN layers are followed by 1 multi-head attention layer. In particular, each GDN head uses GDN (Yang et al., 2025a) extended with negative eigenvalues (Grazzi et al., 2025):

Definition 1: GDN with Negative Eigenvalues (Schlag et al., 2021; Yang et al., 2025a; Grazzi et al., 2025)

Let d be the head dimension. For each token t , we are given $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^d$, $\mathbf{v}_t \in \mathbb{R}^{2d}$, and $\alpha_t, \beta_t \in (0, 1)$, with $\|\mathbf{k}_t\| = 1$. The initial state \mathbf{S}_0 is the 0 matrix. The state $\mathbf{S}_t \in \mathbb{R}^{2d \times d}$ and head output $\mathbf{y}_t \in \mathbb{R}^{2d}$ are updated via

$$\begin{aligned}\mathbf{S}_t &= \mathbf{S}_{t-1}\alpha_t(\mathbf{I} - 2\beta_t\mathbf{k}_t\mathbf{k}_t^\top) + \mathbf{v}_t\mathbf{k}_t^\top \\ \mathbf{y}_t &= \mathbf{S}_t\mathbf{q}_t.\end{aligned}$$

GDN extends the earlier DeltaNet (Schlag et al., 2021; Yang et al., 2024c) by adding the decay factor $\alpha_t \in (0, 1)$ on the state update. The negative eigenvalue extension replaces β_t from the original GDN with $2\beta_t$. Despite its simplicity, this change has important consequences for the expressive power of GDN (Grazzi et al., 2025), and we therefore adopt it—in contrast to the GDN architecture used by Qwen-3-Next (Qwen Team, 2025), Kimi Linear (Kimi Team, 2025), and Qwen 3.5 (Qwen Team, 2026).

A GDN head fits seamlessly into the overall transformer architecture from Olmo 3 since we can use the standard query, key, and value projections as inputs and treat \mathbf{y}_t as the per-head output. It is straightforward to construct a hybrid model by replacing entire attention layers with GDN layers—substituting each attention head with a GDN head. In particular, we replace the sliding-window attention (SWA) layers (75% of layers) from Olmo 3 in this way, resulting in a 3:1 hybridization ratio. In GDN, the value vector is typically twice the length of that in a transformer, and β_t requires a few extra parameters to compute. Thus, we slightly adjust the shape of the model to match Olmo 3 in parameter count and training throughput (see Section 2.2).

Why GDN? As we discuss in Section 3, GDN with negative eigenvalues, unlike linear attention (Katharopoulos et al., 2020) or Mamba (Gu and Dao, 2024), can express state tracking problems that transformers cannot. Moreover, in Section 3, we prove that hybrid models with attention and GDN layers can express more than purely attention or purely GDN models. Complementing these expressivity results, we show empirically in Section 4.1 that hybrid models with GDN exhibit favorable scaling properties relative to transformers, and provide a theoretical explanation for why their additional expressive power should translate into more efficient scaling (Section 4.2).

Another important advantage of Olmo Hybrid’s GDN layers over Olmo 3’s SWA layers is the reduced per-layer inference state size, as shown in Table 1.

Table 1 Per-layer inference state size comparison under the Olmo 3 configuration stored in fp16.

Layer Type	Elements	FP16 Size	vs. GDN
Multi-Head Attention (32K seq, 32 KV heads, $d_h=128$)	268.4M	512 MiB	485×
Grouped-Query Attention (32K seq, 8 KV heads, $d_h=128$)	67.1M	128 MiB	121×
Grouped-Query SWA (4096 window, 8 KV heads, $d_h=128$)	8.39M	16.0 MiB	15.2×
Olmo Hybrid GDN (30 heads, $d_k=96$, $d_v=192$)	0.55M	1.05 MiB	—

2.2 Training Overview

Pretraining. Overall, Olmo Hybrid aims to use the same hyperparameters and training configuration as Olmo 3 to maximize comparability, though we made a few small changes where warranted. In particular,

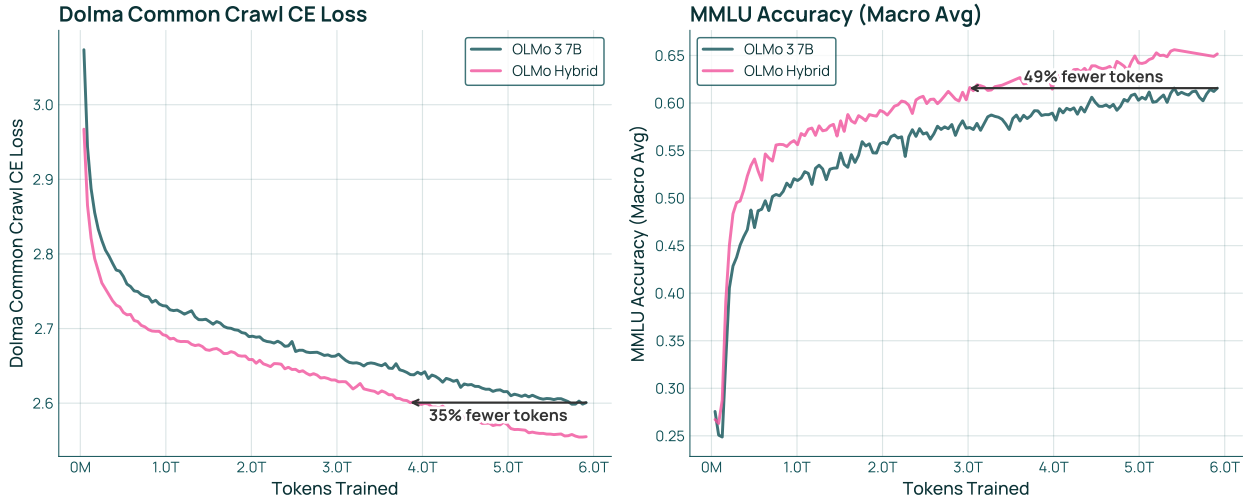


Figure 1 Olmo Hybrid 7B is more efficient than OLMo 3 7B during pretraining, reaching the same Common Crawl loss in 35% fewer tokens and the same MMLU accuracy using 49% fewer tokens (and thus also 35% and 49% fewer FLOPs, respectively).

since GDN has more parameters than a transformer with the same hyperparameters, we reduced the number of heads in Olmo Hybrid from 32 to 30 and set the key and query head dimension to 96 and the value head dimension to 192. This yielded an Olmo Hybrid 7B model with 7.0B parameters (compared to 6.8B for Olmo 3) that closely matched—in fact, slightly outperformed—Olmo 3 7B in early training throughput benchmarks. We will refer to this model as **Olmo Hybrid 7B**. We also made a few other minor changes to the hyperparameters and training configuration from Olmo 3; see Section A.1 for more details.

Mid-Training. Mid-training largely follows the procedure from Olmo 3 (Olmo Team, 2025): adaptation on the mid-training data used for Olmo 3 32B (light filtering applied to the mid-training mixture for Olmo 3 7B) followed by long-context extension. We used a doubled mid-training batch size based on a refined understanding of the interplay between learning rate and batch size since training Olmo 3 (Merrill et al., 2025). As for Olmo 3 32B, we conduct two separate mid-training runs on different 100B token subsets of Dolma 3 Dolmino Mix and merge the final checkpoints. For long-context extension, we experimented with the YaRN methodology (Peng et al., 2024) used for Olmo 3, as well as dropping RoPE entirely (DroPE, Gelberg et al., 2025). DroPE removes RoPE after mid-training, eliminating the extrapolation bottleneck imposed by rotation frequencies learned at shorter context lengths. We hypothesize that DroPE is well suited to hybrid architectures because the GDN layers already provide implicit positional encoding through their recurrent structure, reducing the model’s reliance on RoPE. While we chose DroPE for the released Olmo Hybrid, we evaluate Olmo Hybrid with both methods for comparability (see Table 3). We run long-context extension on 100B tokens of Dolma 3 Longmino Mix.

As a proof of concept, we also apply the Olmo 3 post-training recipe to Olmo Hybrid in Section 5.3.

2.3 Olmo Hybrid Results

Baselines. To evaluate the performance of hybrid models, we focus on the largely clean comparison between Olmo 3 and Olmo Hybrid at a large scale of 7B parameters and 6T tokens. Later, in Section 4.1, we complement this with extensive fully controlled experiments between transformers and hybrid models at smaller scale, as well as, in Section 5.2, comparisons against other released (but not perfectly comparable) transformers, linear RNNs, and hybrid models.

Pretraining Efficiency. As shown in Figure 1, Olmo Hybrid is more compute- and data-efficient than Olmo 3, reaching the same Common Crawl loss and MMLU accuracy as Olmo 3 7B with significantly fewer training tokens. By the end of the 6T-token training run, this increased efficiency of Olmo Hybrid translates to gains

Model	Base Aggregate Scores					Held-out Scores			
	Math	Code	MC _{STEM}	MC _{Non-STEM}	GenQA	LBPP	BBH	MMLU Pro	DM Math
Olmo 3 7B Pretrain	23.3	19.6	64.0	71.9	68.5	6.3	48.6	31.2	15.7
Olmo Hybrid 7B Pretrain	27.0	17.1	67.4	75.5	66.8	2.9	52.2	35.7	13.2
Olmo 3 7B Midtrain	59.8	32.1	67.3	78.2	71.3	17.8	65.9	36.6	23.8
Olmo Hybrid 7B Midtrain	61.2	32.9	70.9	81.3	73.6	17.7	68.6	43.1	23.7
Olmo 3 7B LC	54.6	30.9	66.2	78.2	72.5	17.7	64.0	37.2	23.6
Olmo Hybrid 7B LC	55.1	32.4	70.0	80.4	72.9	16.8	65.2	41.7	23.4

Table 2 Base model evaluations across the training pipeline for Olmo Hybrid vs. Olmo 3. After mid-training, Olmo Hybrid outperforms Olmo 3 7B across every evaluation domain. On evaluations held out from Olmo 3 development (right), Olmo Hybrid outperforms Olmo 3 on MMLU Pro and BBH but loses slightly on LBPP and DM Math. Overall, the results suggest that Olmo Hybrid 7B generally outperforms Olmo 3 7B.

Model	LC Method	RULER Scores				
		4k	8k	16k	32k	64k
Olmo 3 7B	YaRN	95.8	89.3	83.2	78.9	70.9
Olmo Hybrid 7B	YaRN	92.8	91.3	90.0	84.7	76.9
Olmo Hybrid 7B	DroPE	92.2	89.8	88.4	86.2	85.0

Table 3 Results for Olmo Hybrid after long-context tension with YaRN and DroPE compared against Olmo 3. Both methods outperform Olmo 3 7B at long sequence lengths, with the gains from DroPE being particularly notable.

on both metrics compared to Olmo 3. This pretraining efficiency is a compelling fundamental advantage of hybrid models over transformers. Additional pretraining efficiency curves across 6 downstream benchmarks are shown in Appendix Figure 15.

Standard Benchmarks. In Table 2, we compare Olmo Hybrid and Olmo 3 7B across different stages of pretraining and mid-training on domain-specific and held-out evaluation splits from OLMOBASEEVAL. The final pretrained Olmo Hybrid checkpoint outperforms Olmo 3 7B on math (+4.3%), STEM MC (+3.4%), and non-STEM MC (+4.4%), with smaller degradations in code (-2.5%) and general QA (-2.3%). However, after mid-training, Olmo Hybrid outperforms Olmo 3 7B across every evaluation domain, and these performance gains persist after long-context extension. Additionally, we compare Olmo 3 and Olmo Hybrid on evaluations held out from the Olmo 3 development process. On these held-out evaluations, we find gains on MMLU Pro (Wang et al., 2024) (+4.5%) and BBH (Suzgun et al., 2022) (+1.2%) but degradations on LBPP (Matton et al., 2024) (-1.1%) and DM Math (Saxton et al., 2019) (-0.2%). Taken together, the domain-specific and held-out evaluations point to strong performance of the Olmo Hybrid 7B base model relative to Olmo 3 7B.

Long-Context Capabilities. Additionally, after long-context extension (Olmo Team, 2025), we find that Olmo Hybrid shows substantial gains on RULER (Hsieh et al., 2024), a standard long-context benchmark, over Olmo 3. We adapt Olmo Hybrid for long context using both the YaRN methodology (Peng et al., 2024) from Olmo 3 (Olmo Team, 2025) and the more recent DroPE method (Gelberg et al., 2025). As shown in Table 3, Olmo Hybrid 7B outperforms Olmo 3 7B at long RULER lengths with both YaRN and DroPE. We attribute these gains on long-context tasks to the presence of linear RNN layers, in line with existing findings in the literature (Yang et al., 2025b; Ren et al., 2025).

Comparison to Open-Weight Models. Beyond the controlled comparison between Olmo Hybrid and Olmo 3, we also benchmark Olmo Hybrid against other open-weight models of similar parameter count. As shown in Figure 2, Olmo Hybrid is on the Pareto frontier for performance on OLMOBASEEVAL as a function of training compute among open-weight dense models. We detail baseline architectures and present full results in §5.2.

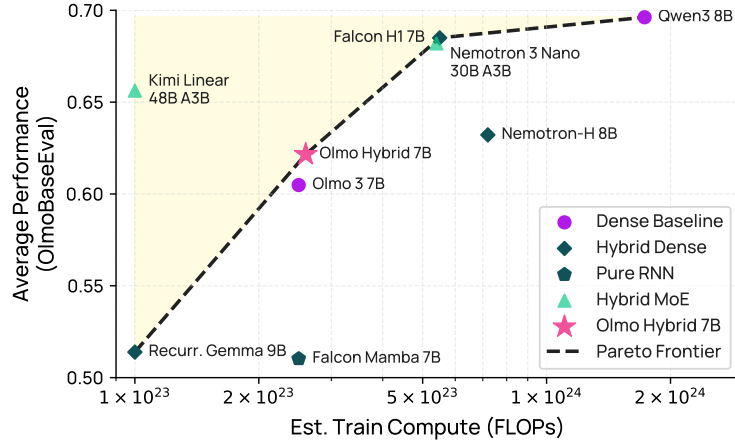


Figure 2 Compute-performance tradeoff of open-weight hybrid, RNN, and transformer base models on the average of OLMOBASEVAL task suites. Olmo Hybrid 7B is on the Pareto frontier of open-weight dense models. Training compute was estimated using the $C = 6ND$ heuristic from Kaplan et al. (2020), with reported token and parameter counts. Per-benchmark results are reported in Table 6. While theoretical compute for MoE-based SSMS is low, effective training efficiency can be roughly 50–80% of a dense model (Rajbhandari et al., 2022), so we draw our frontier over dense models only. For this plot, we show only models obtaining $>50\%$ average performance on OLMOBASEVAL.

3 Expressive Power of Hybrid Models

This section considers the benefits of hybrid models over transformers for *expressive power*, i.e., the class of computational tasks that each architecture can represent. We show that hybrid models can express tasks beyond both transformers and GDN. Later, we return to explaining how this theoretical expressivity advantage could translate to better compute and data efficiency during pretraining compared to transformers (Section 4.1).

While no comprehensive theory of deep learning models exists, theory is advanced enough to provide a useful conceptual framework for understanding the practical limitations of architectures and how to address them. In particular, our motivation for hybrid models takes *expressivity* as a guiding principle:

Expressivity Thesis (Merrill, 2025, Section 5.1)

A deep learning architecture should be made as expressive as possible—being able to represent as many naturalistic problems as possible theoretically—while remaining trainable and scalable for pretraining.

In other words, this perspective calls for designing a highly *flexible* model whose hypothesis class contains as many subtasks as possible that could conceivably be reflected in naturalistic data, while remaining trainable at scale (cf. the Bitter Lesson; Sutton, 2019). Rather than constraining the hypothesis class, we trust the optimizer to find the right fit to the data. This perspective contrasts with classical machine learning wisdom that expressivity and inductive bias are at odds (the bias-variance tradeoff; Mitchell, 1980; Vapnik, 1991; Geman et al., 1992). However, recent work suggests that deep learning methods have an implicit bias toward simplicity that prevents highly expressive models from overfitting (Wilson, 2025). With this in mind, the expressivity thesis outlined above asserts that we should focus on making our architecture maximally expressive without worrying about the impact on inductive bias, as long as the architecture satisfies standard trainability constraints, i.e., the network must be differentiable and signals must propagate across layers during the forward and backward passes (cf. Yang et al., 2024a; Dey et al., 2025).

Beyond trainability, the other major constraint that competes with expressivity is scalability. In general, there is a fundamental tradeoff between expressive power and the degree to which the model can process data efficiently during training. For example, if we want our architecture to exactly express NP-complete problems, it would not be possible to process text with it efficiently (assuming $P \neq NP$). Even if we only require the model to express all tasks in P at training time, the model still could not be *parallelized* effectively over long sequences (assuming $NC \neq P$; Greenlaw et al., 1991), which would preclude scaling training to

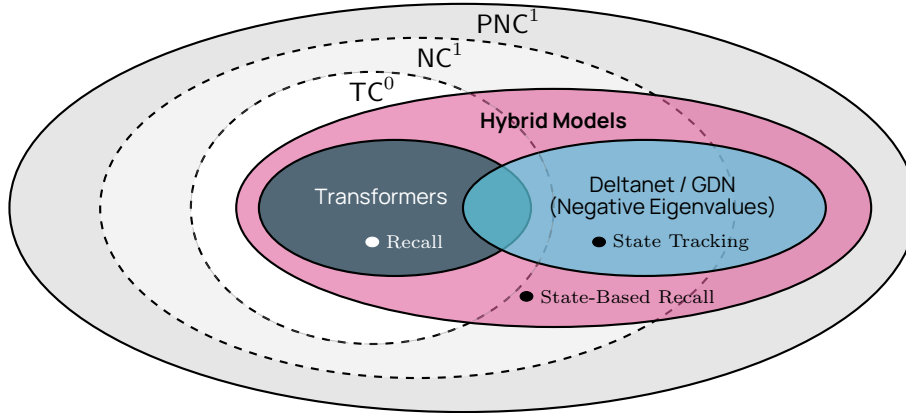


Figure 3 Expressive power of transformers, linear RNNs, and hybrid models relative to circuit complexity classes. Dashed lines represent unproven but *conjectured* separations between classes (e.g., $TC^0 \neq NC^1$). Notably, transformers can express recall (Section 3.1), and DeltaNet (or GDN) with negative eigenvalues can express state tracking (Section 3.2) (Grazzi et al., 2025). Hybridizing gives both capabilities, and we prove that it also unlocks state-based recall, a problem that neither model can express on its own (Section 3.3). With the addition of padding tokens, transformers can express exactly the class TC^0 , whereas hybrid models can capture all of NC^1 , which enables solving boolean formula evaluation (Section 3.4).

massive amounts of data. Thus, our goal in architecture design is not to blindly increase expressivity as much as possible, but to do so while preserving parallelism. A good architecture is one that pushes the frontier of the **expressivity-parallelism tradeoff** (Merrill and Sabharwal, 2023; Liu et al., 2026) as much as possible within fundamental, complexity-theoretic limits.

Existing work has revealed that transformers do not sit on the frontier between expressivity and parallelism: more expressive models exist that are essentially just as parallelizable in practice. In particular, hybrid models can achieve more expressivity while maintaining similar levels of scalability, which we justify both with prior work and new theoretical results. Our core results are summarized in Figure 3. Linear RNNs and transformers have complementary strengths: while linear RNNs like GDN can express state tracking problems beyond the capabilities of transformers, transformers surpass linear RNNs at recall. Each architecture can be trained at scale, but each also has its own expressivity limitations, motivating *hybrid* models that inherit the best properties from each. Moreover, we prove new theoretical results establishing that hybrid models have expressivity advantages *beyond* both pure transformers and pure linear RNNs.

3.1 Background: Expressive Power of Transformers

As the go-to LM architecture, transformers have received significant theoretical treatment investigating what problems they can and cannot solve (Strobl et al., 2024). Despite their empirical success, it has become clear that, when used as next-token predictors, transformers can only express a restricted set of parallelizable computational problems—formally, those that lie within the complexity class TC^0 (Merrill and Sabharwal, 2023; Chiang, 2025). Under standard complexity conjectures, this implies transformers of fixed depth cannot express *inherently sequential* computational problems over arbitrary sequence lengths. This includes many problems like graph connectivity (NL-complete) and computing reward in a Markov decision process (P-complete), but, in particular, it includes the fundamental problem of *state tracking*.

Transformers are Limited at State Tracking. A particularly prominent example of a problem not expressible by transformers (assuming $TC^0 \neq NC^1$) is *state tracking* (Liu et al., 2023a; Merrill et al., 2024, Section 3). State tracking is the task of mapping a world state $x \in X$ and sequence of actions $\delta_1, \dots, \delta_n \in \Delta$ to the updated world state after applying the actions sequentially, e.g., $(\delta_n \circ \dots \circ \delta_1)(x) \in X$, where both X, Δ are finite sets and composition over Δ is associative.¹ One natural example of state tracking is chess, where X is

¹State tracking can be described in many equivalent ways, e.g., the word problem in a finite monoid, recognizing a regular language, or simulating a deterministic finite automaton (cf. Merrill et al., 2024, Section 3).

<pre>a, b, c, d, e = range(5) a, c = c, e ... # n lines assert a == _ # 0 to 4</pre>	<pre>bits = [0, 1, 0, 0, ...] # m bits a = 36 assert bits[a] == _ # 0 or 1</pre>
--	--

Figure 4 Code evaluation contexts where predicting the next token requires solving state tracking (left) and recall (right). As the number of lines n grows, fixed-depth transformers cannot represent state tracking, assuming $TC^0 \neq NC^1$ (Merrill et al., 2024). As the number of bits m grows, RNNs with sub-linear precision cannot solve recall because of their bounded state (Arora et al., 2024b; Jelassi et al., 2024). Hybrid models can represent both problems.

```
bits = [0, 1, 0, 0, ...] # m bits
a, b, c, d, e = 36, 23, 12, 2, 56 # 0 to m - 1

a, c = c, e
... # n lines

assert bits[a] == _ # 0 or 1
```

Figure 5 A code evaluation context where predicting the next token requires solving state-based recall. As n increases, the task becomes inexpressible by transformers because the variable states cannot be tracked (assuming $TC^0 \neq NC^1$). As m grows, the task becomes inexpressible by RNNs because recall into the bit array requires more memory than their bounded state can hold. There exists a simple hybrid model that can solve the task robustly for any value of n and m .

the set of all board states and Δ is the set of all moves. Another example is the “shell game”: the problem of composing swaps (transpositions) over five objects. The complexity of state tracking depends on the algebraic structure of the transition monoid Δ : in the hardest case (capturing the shell game and certain notations of chess; Merrill et al., 2024), state tracking is NC^1 -complete. It follows that these instances of hard state tracking cannot be expressed by fixed-depth transformers assuming the complexity conjecture $TC^0 \neq NC^1$, even though variants of these problems could conceivably manifest as subproblems of next-token prediction over natural language data. Intuitively, the problem arises from the fact that attention cannot aggregate information over the sequence of state updates in a way that is fully sensitive to their order (each attention head could look to the last token and apply an individual update, but then the depth of the network would grow with the sequence length).

3.2 Background: Expressive Power of Linear RNNs

In contrast to transformers, classical nonlinear RNNs can naturally express state tracking by using their hidden vector as a representation of the intermediate state and applying each transition one at a time (e.g., Merrill, 2019, Theorem 3.1). A fundamental question is whether linear RNNs, with their linear recurrent update, can still express sequential state updates in a similar way.

Linear RNNs Can Track State. It turns out that some (but not all) linear RNNs, despite being efficiently parallelizable, can express sequential state tracking. Merrill et al. (2024) showed how early linear RNN architectures like S4 (Gu et al., 2022) and Mamba are limited in complexity to TC^0 like transformers, and thus cannot express state tracking under standard conjectures. However, Merrill et al. (2024) also showed how, by making the transition matrix *non-diagonal* and *time-dependent*, it is possible to construct a linear RNN that can express NC^1 -complete state tracking problems. Along these lines, recent linear RNN architectures such as DeltaNet (Yang et al., 2024c; Grazzi et al., 2025), RWKV-7 (Peng et al., 2025), and PD-SSM (Terzic et al., 2025) have incorporated more complex transition matrix parameterizations that unlock additional expressive power for state tracking. In the case of DeltaNet or GDN, extending the transition matrix to have negative eigenvalues (Definition 1) is important for state tracking (Grazzi et al., 2025). Intuitively, negative eigenvalues are important because they allow DeltaNet to express swap-like dynamics that alternate between two states, as shown in Figure 6. On empirical state tracking evaluations (such as the A_5 word problem), these more expressive parameterizations lead to clear improvements over transformers. In other related work, Sarrof

$$\mathbf{I} - \mathbf{k}_t \mathbf{k}_t^\top = \mathbf{I} - \frac{1}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \qquad \mathbf{I} - 2\mathbf{k}_t \mathbf{k}_t^\top = \mathbf{I} - \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

(a) Without the negative eigenvalue extension (cf. Definition 1), we get eigenvalues $\lambda_1 = 1$ and $\lambda_2 = 0$.

(b) With the extension, GDN implements a “swap” operator with eigenvalues $\lambda_1 = 1$ and $\lambda_2 = -1$.

Figure 6 Let $\mathbf{k}_t^\top = (1, -1)/\sqrt{2}$; crucially, $\|\mathbf{k}_t\| = 1$. Multiplying the low-rank term in the GDN update by 2 allows the transition matrix to have a negative eigenvalue, meaning it can implement dynamics that alternate between states (Grazzi et al., 2025). This is useful for expressing state tracking tasks like parity and the A_5 word problem.

et al. (2024) show advantages of linear RNNs over transformers on star-free regular languages, a simpler type of state tracking compared to A_5 , and Merrill et al. (2026) show that DeltaNet and other linear RNNs like RWKV-7 are upper bounded by PNC¹ and can solve PNC¹-complete problems.

Linear RNNs are Limited by Recall. While linear RNNs have an expressivity advantage over transformers on state tracking tasks, this advantage comes at a cost: they are limited on recall-heavy tasks due to their bounded state. In particular, the fixed-size hidden state of linear or nonlinear RNNs means they struggle on tasks involving copying or recalling tokens from the context (Jelassi et al., 2024; Arora et al., 2024b). In-context recall mechanisms have been suggested to be important for language modeling (Olsson et al., 2022), and, indeed, Arora et al. (2024a) argue that most loss degradation of linear RNNs relative to transformers can be attributed to in-context recall. Similarly, Akyürek et al. (2024) argue that transformers outperform pure RNNs at in-context learning, likely due to recall abilities. Additionally, in-context recall is important in constructions for Turing completeness when models are augmented with chain of thought (Merrill and Sabharwal, 2024; Wen et al., 2025a). Thus, for several reasons, it is important for RNN-based models to incorporate some mechanism for recall, providing a strong motivation for hybrid transformer-RNN architectures.

3.3 Hybrid Models are More Than the Sum of Their Parts

Section 3.2 demonstrates that linear RNNs like DeltaNet have a key expressivity advantage over transformers on state tracking tasks. On the other hand, they are limited on recall-heavy tasks due to their bounded state size. From this perspective, hybrid models that mix layer types offer a natural way to build a model that can express both state tracking and recall while remaining scalable (Yang et al., 2025b; Mohri, 2026). Going beyond this, we now show that hybrid models are, in fact, *more expressive* than either transformers or linear RNNs in isolation, under standard complexity conjectures. In particular, we will first show the expressivity advantage of hybrid models on the minimal synthetic task of state-based recall:

Definition 2: State-Based Recall (Figure 5)

The input is a string x, p, π , where $x \in \{0, 1\}^n$ is a bitstring, $p_1, \dots, p_5 \in [1, n]$ are “pointers” into the bitstring, and π_1, \dots, π_n is a sequence of transpositions (swap operations) over $[1, 5]$. Define the permuted pointer values q as $q = (\pi_n \circ \pi_{n-1} \circ \dots \circ \pi_1)(p)$. The output is x_{q_1} .

Intuitively, state-based recall is designed to require *composing* state tracking and recall, rather than just one capability. State-based recall can be naturally instantiated as a subtask of *evaluating code*, which requires both tracking the state of variables and using their values for memory accesses (e.g., indexing into a list). Figure 5 shows an example of how state-based recall might appear within code language modeling. This gives an intuition for how the additional expressivity of hybrid models could be relevant for tasks involving code evaluation. We now show formally that, since it requires the composition of state tracking and recall, state-based recall is solvable by hybrid models, but not pure transformers or GDN models:²

²Our formal models of transformers and RNNs operate under the common *log-precision arithmetic* assumption (Merrill and Sabharwal, 2023; Merrill et al., 2024), where all internal arithmetic is allowed to use $O(\log n)$ bits for input sequences of length n . All our inexpressibility results easily carry over to bounded-precision models. Theorem 1 holds also for poly-precision transformers (Chiang, 2025) and can be extended to sub-linear-precision RNNs.

Theorem 1: State-Based Recall Separation

There exists a hybrid model (GDN with negative eigenvalues + averaging-hard attention) that solves state-based recall, with just one alternation between layer types, in either order. In contrast, no transformer or RNN can express this problem, assuming $\text{TC}^0 \neq \text{NC}^1$ for transformers.

Proof Sketch. A hybrid model can solve state-based recall in two ways. One way is to first use GDN layers to compose the transpositions over pointers (an example of NC^1 -complete state tracking) and then retrieve the value at p_1 using attention. The other way is to first use attention heads to retrieve the values of p_1, \dots, p_5 and then compose transpositions over the values. Thus, hybrid models with a single alternation of layer types in either order can express the task. On the other hand, full transformers and GDN each lack the ability to express one of the pieces, and thus cannot express state-based recall. We defer a rigorous proof of these negative results to Section B. \square

Since state-based recall is expressible with GDN before attention or vice versa, it follows that alternating layers in either order also unlocks additional expressivity for hybrid models relative to full attention or full GDN. It is an open question whether multiple alternations between layer types buy more expressivity than having just one alternation.

3.4 Expressive Power of Padded Hybrid Models

The interaction between attention and GDN layers naturally allows hybrid models to solve state-based recall, providing a minimal example of an expressivity gain over pure transformers and linear RNNs (under standard conjectures). Moreover, we now show that it also provides more general expressivity benefits for hybrid models: in the presence of padding tokens, hybrid models can express *every* problem in NC^1 , which includes many NC^1 -complete problems that are beyond the capabilities of padded transformers (which remain in TC^0). In particular, our results imply that, unlike padded transformers, padded hybrid models can evaluate boolean formulas, even though, at first glance, it is non-obvious why the interaction of attention and recurrence should enable this.

Before presenting our results, we briefly introduce *padding tokens* and their relevance to theoretical analysis of expressivity. Padding tokens (Goyal et al., 2024; Pfau et al., 2024) are simply blank tokens (\square) appended to a model’s input context: i.e., a model with n^c padding takes as input $w\square^{|w|^c} \in \Sigma^*$ rather than just $w \in \Sigma^*$. Padding tokens are interesting for theoretical analysis because adding padding tokens leads architectures to subsume (or exactly correspond to) natural complexity classes in expressivity. Without padding tokens, showing that an entire circuit complexity class neatly lower bounds an architecture is not always possible, in large part because circuit complexity classes like TC^0 and NC^1 allow arbitrary polynomial size computation, whereas models leverage a computation graph of size $O(n^c)$ for some fixed c . Adding padding tokens can close the gap between poly-size circuit classes and neural sequence models by allowing them to expand the size of the model’s computation without adding new parameters (Li et al., 2024; Merrill and Sabharwal, 2025; London and Kanade, 2025).

It is already established that, with padding tokens, the exact class of problems expressible by transformers has a natural characterization. In particular, while averaging-hard-attention transformers without padding tokens are bounded within TC^0 (Merrill and Sabharwal, 2023), with polynomial padding, the languages expressible by such transformers are *exactly* TC^0 (Merrill and Sabharwal, 2025):

Theorem 2: Padded Transformers (Merrill and Sabharwal, 2025)

With polynomial padding tokens, fixed-depth transformers with averaging-hard attention recognize exactly FO-uniform TC^0 .

We give a comparable result for hybrid models with polynomial padding, showing they capture *all* of NC^1 , a complexity class thought to be larger than TC^0 :

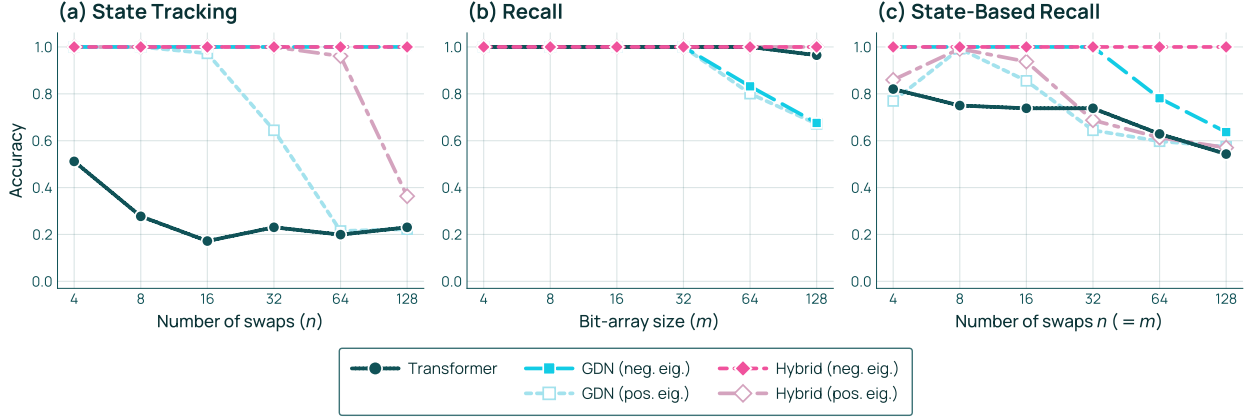


Figure 7 Synthetic task results. **(a)** State tracking accuracy vs. number of updates n : linear RNN and hybrid remain near-perfect, while the transformer falls quickly. **(b)** Recall accuracy vs. bit-array size m : transformer and hybrid maintain perfect recall, while the linear RNN degrades for large m . **(c)** State-based recall accuracy vs. number of swaps n ($m = n$): the hybrid remains robust, while both the transformer and linear RNN degrade for large m, n .

Theorem 3: Padded Hybrid Models

With polynomial padding tokens, fixed-depth hybrid models (averaging-hard attention + GDN with negative eigenvalues) can recognize any language in FO-uniform NC^1 .

Proof Sketch. We use the surprising classical result that any problem in NC^1 can be reduced via a first-order (FO) formula to composing transpositions over 5 elements (Barrington, 1986). Padded attention layers can express an arbitrary FO reduction, and GDN can implement transposition composition. Thus, this decomposition allows solving any problem in NC^1 with a padded hybrid model. Full proof in Section B. \square

In contrast to Theorem 1, which establishes an expressivity advantage for hybrid models on a specific problem, Theorem 3 establishes that padded hybrid models subsume an entire complexity class (NC^1) that, under standard complexity conjectures, is more powerful than the class padded transformers correspond to (TC^0). Under the conjecture that $\text{TC}^0 \neq \text{NC}^1$, any NC^1 -complete problem can be expressed by a padded hybrid model but not by a padded transformer. One interesting problem in this regime is **boolean formula evaluation**. Let ϕ be a boolean formula over $\{\{0, 1\}, \vee, \wedge\}$. The evaluation problem takes as input ϕ serialized in Polish notation, and the output is the value of ϕ . Since formula evaluation is NC^1 -complete (Buss, 1987), we obtain:

Corollary 3.1: Boolean Formula Evaluation Separation

For some c , there exists a hybrid model (averaging-hard attention + GDN with negative eigenvalues) that solves boolean formula evaluation with n^c padding tokens. On the other hand, assuming $\text{TC}^0 \neq \text{NC}^1$, no transformer or RNN can solve boolean formula evaluation even with n^c padding tokens, for any c .

In contrast to state-based recall, it is not obvious at first glance that boolean formula evaluation should be expressible via the interaction of attention and recurrence. However, Corollary 3.1 shows that it can be, at least for padded models, and moreover that, under standard complexity conjectures, it could not be solved with just one of these primitives. Corollary 3.1 can be restated to fold the padding into the problem definition itself: *unpadded* hybrid models can solve *padded* formula evaluation, but neither transformers nor linear RNNs can. Finally, the only property of boolean formula evaluation leveraged to obtain Corollary 3.1 is that it is NC^1 -complete. Thus, a similar result follows for any NC^1 -complete problem. It is an open question whether a similar boolean formula evaluation result might be obtained for unpadded hybrid models.

3.5 Synthetic Evaluations

To empirically validate the expressivity tradeoffs from Sections 3.1 to 3.3, we train three primary causal LMs—a standard transformer, a linear RNN (GDN with negative eigenvalues), and a hybrid model mixing GDN blocks with full attention—on synthetic state tracking, recall, and state-based recall tasks. We additionally evaluate no-negative-eigenvalue ablations of the linear RNN and hybrid model. Each task is generated online from short code-like templates (cf. Figures 4 and 5) and framed as a next-token prediction task where the final token requires evaluating the program correctly. We report next-token accuracy on the final answer token and vary task difficulty by increasing the number of required state updates and/or the size of the stored bit-array. For the state-tracking and state-based recall tasks, we additionally train on execution traces that include intermediate state-reveal checks in the form of assert statements, and we include these tokens in the language-modeling loss following Siems et al. (2026). Complete experimental details (model hyperparameters, optimization, curricula, and ablation runs) are provided in Section C.

State Tracking. To evaluate state tracking, we use the code evaluation task in Figure 4, where the context applies a sequence of state updates and the model must predict the value of a queried component of the final state. As shown in Figure 7, the GDN-based linear RNN and the hybrid remain near-perfect across all tested lengths, while the transformer drops rapidly as the update sequence length increases, consistent with the theoretical limitation of fixed-depth attention on state tracking (Section 3.1). The no-negative-eigenvalue ablations substantially weaken this behavior: the linear RNN without negative eigenvalues falls to 0.21484 and 0.22266 at $n = 64$ and 128, and the corresponding hybrid falls to 0.36328 at $n = 128$. In this implementation, negative eigenvalues appear important for robust recurrent state tracking, consistent with the findings from Grazzi et al. (2025).

Recall. To evaluate pure recall, we use the code evaluation task illustrated in Figure 4, where predicting `bits[idx]` requires retrieving a single entry from a long list of bit assignments. As shown in Figure 7, the transformer remains near-perfect and the hybrid remains perfect across the tested context sizes, while the linear RNN degrades as the number of bits grows. The no-negative-eigenvalue ablation leaves the hybrid unchanged and only slightly worsens the linear RNN on the hardest settings (0.80078 at $m = 64$ and 0.67188 at $m = 128$, compared with 0.83203 and 0.67578 with negative eigenvalues). This matches the theoretical expectation that linear RNNs should struggle with recall due to their bounded-size state (Section 3.2).

State-Based Recall. To test the composition of state tracking and recall, we use the state-based recall task in Figure 5, where state updates transform a set of pointers that are subsequently used to index into a bit array. Theoretically, Theorem 1 predicts hybrid models can express state-based recall while transformers and RNNs cannot under standard complexity conjectures; we now test empirically whether a separation is observed when these models are trained on the task. As shown in Figure 7, both the transformer and the pure GDN-based linear RNN degrade as difficulty increases, while the hybrid with negative eigenvalues remains essentially perfect across all tested difficulties. By contrast, the no-negative-eigenvalue hybrid loses this advantage, dropping to 0.57031 at $n = 128$; the corresponding no-negative-eigenvalue linear RNN is similarly weak. This suggests that, in practice, the hybrid’s compositional advantage depends on the recurrent block’s access to negative eigenvalues.

Overall, these results support the theoretical picture while refining the architectural story: transformers excel at recall but struggle with hard state tracking, GDN-based linear RNNs show the opposite pattern, and hybrids with negative eigenvalues inherit both capabilities and remain strong on their composition. Removing negative eigenvalues has little effect on pure recall, but substantially harms state tracking and the composed state-based recall task, in line with theoretical predictions (Grazzi et al., 2025).

3.6 Discussion: Pushing the Expressivity-Parallelism Frontier

Transformers and linear RNNs have complementary strengths from an expressivity perspective. We have shown that hybrid models do not simply inherit the strengths of each architecture; they go beyond both transformers and linear RNNs by expressing tasks that neither architecture alone can. Notably, this expressivity advantage of hybrid models comes despite them retaining theoretical parallelizability to a similar degree as transformers

(Merrill et al., 2026). In other words, hybrid architectures extract strictly more from the level of parallelism at which transformers and linear RNNs operate, in a similar vein to the idea of “leaving less money on the table” and getting more out of the same resources in machine learning (Ligett, 2026). Thus, hybrid models push the expressivity-parallelism frontier for language modeling architectures beyond transformers (cf. beginning of Section 3), yielding fundamentally more expressive models that remain similarly scalable.

4 Scaling Behavior of Hybrid Models

Having established the increased theoretical expressivity of hybrid models over transformers in Section 3, we now turn to evaluating hybrid LMs in practice. A central question is whether the theoretical expressivity guarantees for hybrid models translate to better empirical performance as a function of the parameter count and data invested into a language model.

First, by fitting scaling laws for hybrid models and transformers in a carefully controlled setting (Section 4.1), we establish that hybrid models achieve better scaling efficiency on loss-based pretraining metrics compared to transformers—particularly in scaling with data quantity—consistent with our findings from the Olmo Hybrid pretraining run. Next, drawing on existing theoretical explanations for scaling laws, we present a theoretical argument that increasing an LM’s expressive power should improve its scaling efficiency (Section 4.2). Informally, building on explanations of scaling laws in terms of the multi-task nature of language modeling (Michaud et al., 2023), we argue that increased expressivity can improve scaling because it allows a model to acquire a larger proportion of the discrete computational tasks reflected in the pretraining data. This provides a conceptual explanation for the improved scaling efficiency of hybrid models that we take as a compelling hypothesis for future work to test and develop further.

4.1 Scaling Laws for Hybrid Models

This section presents derived empirical scaling laws for transformers, pure GDN models, and hybrid models that form the backbone of Olmo Hybrid. Empirical scaling laws enable a principled comparison of different architectures, evaluating not only performance at specific scales but also projecting to larger scales. Our results show that hybrid models are both more data-efficient and more compute-efficient than transformers across scales.

Overview. We fit Chinchilla-style scaling laws (Hoffmann et al., 2022) to transformers, pure GDN models, and hybrid GDN–transformer models. We find that the hybrid model has a meaningfully lower data coefficient B , while scaling exponents are statistically indistinguishable across architectures. This aligns with our theoretical analysis in Section 4.2 that increased expressivity should improve scaling efficiency by reducing the data coefficient, which captures the fixed-factor improvement in the data required to reach a target loss. This translates to projected token savings of ~ 1.3 – $1.9\times$ at model sizes from 1B to 70B parameters.

Scaling Laws Formulation. Scaling laws describe the behavior of the language modeling loss as a smooth power law with model size and data budget (Kaplan et al., 2020). Hoffmann et al. (2022) investigate scaling laws with the parametric form

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \quad (1)$$

where the number of parameters N and number of training tokens D are the independent variables. The quantity E is the irreducible loss, i.e., the loss that would be attained with infinite resources. The other fit parameters govern how efficiently loss is reduced when N, D are scaled. The coefficients A and B capture fixed-factor improvements in the resources required to reach a target loss: reducing either by a factor k with fixed E implies that the same target loss can be reached with a k -fold reduction in resources. Finally, the exponents α and β govern the rate at which loss improves with scale, though generally these exponents are not changed much by architecture choices. Comparing these parameters between two architectures provides a principled way to quantify which is fundamentally more compute- or data-efficient.

We now fit scaling laws to evaluate how efficiently hybrid models scale relative to other architectures. We evaluate three architectures—a pure transformer, a pure GDN, and a hybrid GDN architecture with every fourth

layer being a full transformer block—across scales from 60M to 760M parameters, fitting scaling laws to estimate their coefficients and project performance at larger compute budgets. We follow an **isoparams data collection strategy** (Hoffmann et al., 2022): for each model size $N \in \{60\text{M}, 100\text{M}, 190\text{M}, 370\text{M}, 600\text{M}, 760\text{M}, 1\text{B}\}$, we train a model at $0.5\times, 1\times, 2\times, 4\times$, and $8\times$ Chinchilla-optimal tokens (20 tokens per parameter). Naïvely, this would require launching *five* separate runs per architecture and parameter budget. We avoid this by using a WSD-S (warmup–stable–decay with periodic resets) learning rate schedule (Hu et al., 2024; Wen et al., 2025b), which is **token-agnostic**: the learning rate at step t does not depend on the total number of tokens T . This allows us to *reuse* intermediate checkpoints from a single long run to collect loss measurements at different data budgets D . More precisely, we *decay* the learning rate for 5% of the training tokens to 0 at each of the five Chinchilla factors to obtain the trained checkpoint for that Chinchilla factor. Decaying the learning rate at each of the five Chinchilla factors and evaluating the resulting checkpoints yields five (N, D, L) triples from a single training run per architecture and parameter count.

We run this procedure for three models:

1. The transformer baseline based on the Olmo 3 model,
2. A fully linear RNN (GDN), and
3. A hybrid model with a 3 : 1 GDN-to-transformer layer ratio—the architecture that forms the basis of Olmo Hybrid.

We train all architectures under **matched conditions**: identical training data (Olmo 3 32B mix), optimizer settings, batch size scaling, and evaluation. The only difference between runs is the architecture itself.

Model Specification. Rather than matching parameter counts exactly, we match the architectural blueprint—the number of layers, heads, and the hidden dimension—of each model to that of Olmo 3 at each scale. This results in models with different numbers of parameters. We account for these differences by focusing on performance as a function of the training *FLOPs* and fit the scaling laws with respect to the exact number of parameters. The detailed configurations are presented in Table 22.

Fitting and Using Scaling Laws. Given the collected (N, D, L) data triples (35 per architecture; 5 for each of the 7 scales), we fit parametric scaling laws following Approach 3 of Hoffmann et al. (2022): we directly fit $L(N, D) = E + A/N^\alpha + B/D^\beta$ by minimizing the Huber loss of $\log L$ between predictions and data. We model the *validation* loss, computed as the average cross-entropy across 11 held-out evaluation domains: C4 (Raffel et al., 2019), Dolma Books, Dolma Common Crawl, Dolma pes2o, Dolma Reddit, Dolma Stack, Dolma Wiki (Soldaini et al., 2024), ICE (Greenbaum and Nelson, 1996), M2D2 S2ORC (Reid et al., 2022; Lo et al., 2020), Pile (Gao et al., 2020), and WikiText-103 (Merity et al., 2016). Averaging across domains provides a less noisy and more representative signal than any single validation set. Following Hoffmann et al. (2022), we use Huber loss to reduce sensitivity to outliers from training instabilities. The fit is performed jointly over all (N, D, L) triples for each architecture separately. To assess uncertainty in our scaling law estimates, we additionally compute 95% confidence intervals via bootstrap resampling (1,000 iterations).

Free and Fixed-Exponent Fits. We fit scaling laws in two ways. In the *unconstrained* fit, all five parameters (E, A, α, B, β) are free; this provides the most flexible fit that explains the scaling laws but yields wide bootstrap confidence intervals, making per-coefficient comparisons unreliable. We therefore also present a *fixed-exponent* fit in which we fix α and β across all architectures and fit only E, A , and B . Fixing the exponents concentrates the remaining variance onto the efficiency coefficients, enabling more statistically robust comparisons between architectures. Full details on the data used for fitting, the optimization procedure, and the construction of all figures and tables in this section are provided in Section D.2.

Scaling Law Fits and Fit Quality. Figure 8 shows the fitted scaling laws alongside the raw data for all three architectures, plotted against compute, parameter count, and token budget. The curves in all figures are drawn using the coefficients estimated from the unconstrained fit; we use the fixed-exponent fit only when making quantitative comparisons between architectures (see below). Qualitatively, the hybrid and pure GDN curves lie consistently below the transformer curve, indicating lower loss at matched compute or parameter budget; the separation is visible across all three panels. The fits achieve $R^2 \geq 0.998$ for all three architectures,

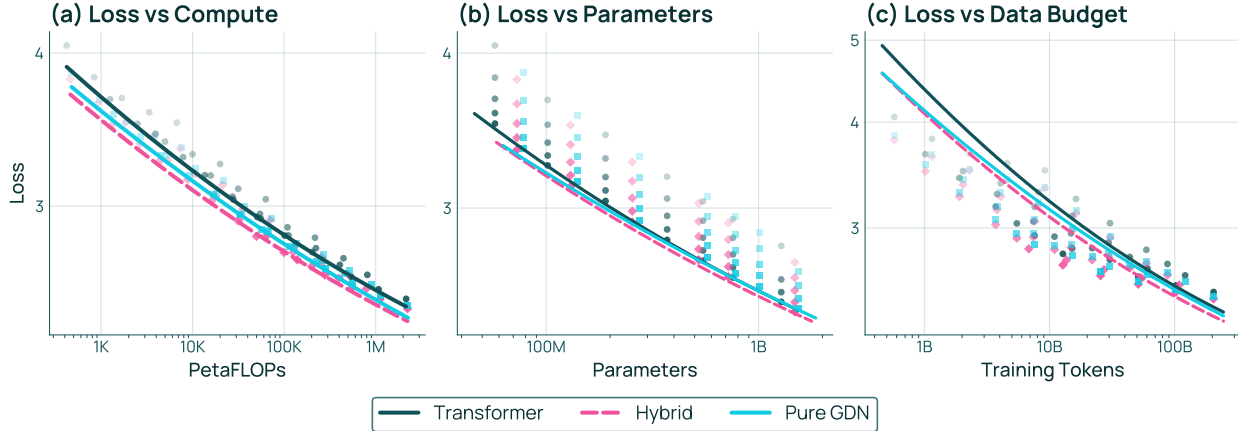


Figure 8 Scaling law fits $L(N, D) = E + A/N^\alpha + B/D^\beta$ for all architectures. Transformer: $\alpha = 0.252$, $\beta = 0.213$, $R^2 = 0.998$. Hybrid: $\alpha = 0.226$, $\beta = 0.219$, $R^2 = 0.999$. Pure GDN: $\alpha = 0.183$, $\beta = 0.227$, $R^2 = 0.999$. **(a)** Loss vs. compute; points represent model checkpoints at different model sizes (shown by opacity) and the bold curve shows the fitted scaling law at the compute-optimal frontier. **(b)** Loss vs. parameter count; points represent checkpoints at different Chinchilla multiples (D/N ratios, shown by opacity) and the fitted curve is evaluated at the largest multiple. **(c)** Loss vs. data budget; points represent checkpoints at different model sizes (shown by opacity) and the fitted curve is evaluated at the largest model size. Loss axes are plotted on a logarithmic scale so that the power-law relationships appear linear.

Architecture	Model	N	D	Observed	Predicted	Error (%)
Transformer	Olmo 3	7B	5.9T	2.17	2.16	0.28
Transformer	Olmo 3	32B	5.5T	2.02	2.05	1.69
Hybrid GDN	Olmo Hybrid	7B	5.5T	2.14	2.14	0.28

Table 4 Scaling law prediction validation against final models. Predicted loss is computed from the fitted law $L(N, D) = E + A/N^\alpha + B/D^\beta$. Extrapolation is generally strong despite differences in the learning rate schedule between our scaling studies and the large-scale training runs.

confirming that the Chinchilla scaling law form describes the observed loss curves well across the full range of scales considered.

The Derived Scaling Laws are Consistent with Pretraining Observations. Table 4 validates the derived scaling laws against *larger* models—the final base 7B Olmo Hybrid as well as the 7B and 32B base Olmo 3 models—and confirms that they generalize well outside the fitting range. The predicted loss for the 7B Olmo Hybrid trained on 5.5T tokens is 2.142, compared to the observed 2.136 (error 0.28%); for Olmo 3 7B (5.93T tokens), the error is similarly 0.28%; and even for Olmo 3 32B—well beyond the ~ 1 B fitting range—the error is only 1.69%. These results establish that the fitted scaling laws are reliable enough to project performance at larger scales.

The Hybrid Model Has a Robustly Lower Data Coefficient. Figure 8 visually suggests that the hybrid model has better scaling efficiency than the transformer, particularly with respect to data. To quantify this, we compare the fitted scaling law parameters across architectures in Figure 9 (see Table 18 in the appendix for full details). While the unconstrained fit allows all five parameters to vary and is thus the most flexible, the resulting CIs are wide and largely overlapping, yielding no statistically robust per-coefficient conclusions. We therefore turn to the fixed-exponent fit to draw robust conclusions: we fix $\alpha = \beta = 0.22$ shared across architectures (the mean of the unconstrained estimates) and refit only E , A , and B . This gives us a clear signal in the *data efficiency* coefficient B : the hybrid’s $B = 83.7$ (CI [80.2, 87.1]) is significantly lower than the transformer’s 94.9 (CI [88.7, 102.0]), with non-overlapping confidence intervals. The parameter coefficient A also slightly favors the hybrid (70.1 vs. 71.8), but the CIs overlap. The scaling exponents in the unconstrained fit are

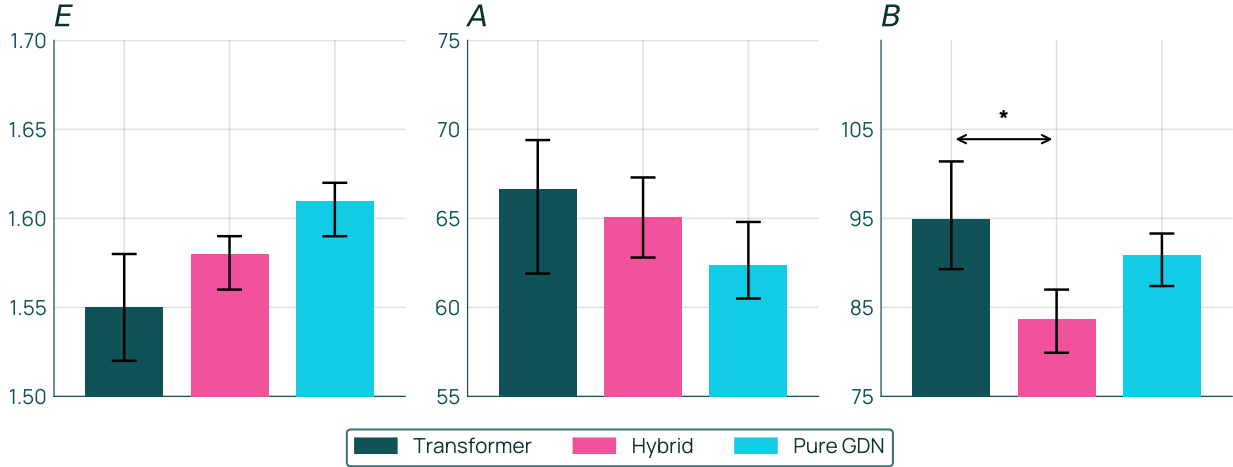


Figure 9 Scaling law coefficients E , A , B with 95% bootstrap CIs for the fixed-exponent fit ($\alpha=\beta=0.22$ shared), which yields tight confidence intervals. The data efficiency coefficient B shows a robust advantage for Hybrid GDN over the transformer (83.7 vs. 94.9, non-overlapping CIs, marked *), while differences in E and A are not statistically conclusive.

statistically indistinguishable across architectures—consistent with the theory (Section 4.2), which predicts that expressivity shifts the efficiency constants without altering the power-law exponents. The primary *robust* advantage of the hybrid model is thus its data efficiency coefficient B , as also predicted by our theoretical results in section 4.2.

Token Savings Grow Steadily with Model Size. The improved scaling trend of hybrid models naturally translates to better scaling of the loss with compute. The projected savings are derived by analytically inverting the fitted scaling law: for a target loss L^* , the number of tokens required by a model with N parameters is $D^*(N) = (B/(L^* - E - A/N^\alpha))^{1/\beta}$, evaluated at each model size using each architecture’s fitted coefficients (see Section D.2 for full details). While the pure GDN achieves the lowest estimated loss at small parameter budgets, the hybrid model achieves better loss at the more typical medium and large scales, where its advantage in the data-efficiency coefficient B becomes the dominant and most reliable signal. Figure 10 visualizes the projected parameter and data savings factors across model sizes for a target loss of 2.474 (the minimum observed training loss); the token requirements are obtained by analytically inverting the fitted scaling law as described in Section D.2. The parameter savings factor (cf. Figure 10(a)) reveals that transformers are more parameter-efficient at larger token budgets, a consequence of the larger scaling factor α . Figure 10(b), in contrast, shows that the *token* savings factor grows steadily with model size, rising from $\sim 1.3\times$ at 1B parameters to $\sim 1.9\times$ at 70B parameters. Concretely, to match a transformer trained at a given scale, one can train a hybrid model of the same size on $\sim 1.3\text{--}1.9\times$ fewer tokens (e.g., $1.68\times$ at 7B, $1.82\times$ at 30B, $1.89\times$ at 70B; see Figure 10(b) and Table 20). The pure GDN model, by contrast, initially requires *more* data than the transformer at the 1B scale ($0.82\times$, i.e., no savings) before catching up and achieving savings of $\sim 1.7\times$ at 70B.

Compute Savings Also Grow Steadily with Model Size. Token savings directly lead to improved compute efficiency. Figure 10(c) visualizes this by showing the compute savings factor needed for reaching a target loss, revealing steadily growing savings. At 10^{22} FLOPs, for example, the hybrid model is projected to achieve a loss of 2.267 compared to 2.308 for the transformer ($\Delta = -0.042$, 95% CI $[-0.14, +0.06]$); full results across compute budgets from 10^{18} to 10^{23} FLOPs are reported in Table 19 in the appendix.

Implications. The derived scaling laws suggest favorable scaling for hybrid models compared to both pure transformers and purely linear RNN models. The most statistically robust finding, from the fixed-exponent analysis, is that the hybrid model has a meaningfully lower data coefficient B (83.7 vs. 94.9 for the transformer, non-overlapping 95% CIs), indicating that hybrid models are more data-efficient learners. The parameter coefficient A also slightly favors the hybrid, though this difference is less statistically clear. In Section 4.2,



Figure 10 Projected savings factors across model scales (target loss = 2.474, selected as the minimum of all observed training losses). **(a)** Parameter savings ($N_{\text{ref}}/N_{\text{arch}}$) vs training tokens: fewer parameters are needed to reach the same loss at a given data budget. **(b)** Data savings ($D_{\text{ref}}/D_{\text{arch}}$) vs model size: fewer training tokens are needed to reach the target loss at a given model size. **(c)** Compute savings ($C_{\text{ref}}/C_{\text{arch}}$, $C \propto N \cdot D$) vs target loss: for each target loss the minimum total compute is found by optimising over model size N ; harder targets (lower loss) are on the right. Dashed vertical lines mark the observed losses of our final production models. The savings factor grows with difficulty for both Pure GDN and Hybrid models. Values above $1 \times$ indicate an advantage over Transformer. Note that estimates at lower loss values (right side of the plot) involve extrapolation beyond the fitting range and are therefore less certain.

we argue that the improved B is consistent with the expressivity-grounded prediction that more expressive architectures can learn a larger proportion of discrete tasks from data, making each training token more valuable. The scaling exponents α and β are statistically indistinguishable across architectures—consistent with the theoretical prediction that expressivity shifts the constant-factor efficiency without altering the power-law exponents. Together, these results suggest that the primary practical benefit of hybrid models for scaling is more efficient use of training data, which is consistent with our pretraining observations (cf. Figure 1) and the projected $\sim 1.9 \times$ data savings at 70B scale (cf. Figure 10(b) and Table 20). We study additional architecture ablations in Section 5.1, where we justify choosing GDN for the linear component of Olmo Hybrid and the specific hybridization strategy.

4.2 Theory: Increased Expressive Power Improves Scaling

In Section 4.1, we saw that the hybrid model scaled more efficiently during pretraining than the transformer, achieving better performance with the same token or compute budget. At first glance, it may be unclear why pretraining efficiency should be related to the greater expressivity of hybrid models (Section 3): after all, our expressivity guarantees imply a binary difference in abilities on synthetic tasks, whereas scaling laws concern smooth improvements on modeling natural-language data. However, we now formalize a plausible explanation for why the greater expressivity of hybrid models should translate to improved scaling behavior. In particular, recent work explains neural scaling laws as emerging from the gradual aggregation of many discrete tasks reflected in language modeling data. Working within this framework, we show that increasing expressivity improves scaling trends (Corollaries 4.1 and 4.2) because it increases the number of discrete tasks that a model can learn on a fixed parameter and token budget. In aggregate, this means that more expressive models can achieve lower loss on the same budget, as shown in Figure 11.

Quantization Model (Michaud et al., 2023). It is well established that language modeling loss decreases smoothly with model size and token budget despite the fact that many properties of language are discrete (Kaplan et al., 2020; Hoffmann et al., 2022). One explanation is that language modeling is fundamentally a multi-task problem: LMs acquire individual tasks discretely, leading to a smooth reduction of aggregate loss

(Hutter, 2021; Arora and Goyal, 2023; Nam et al., 2024, inter alia). In particular, this intuition has recently been formalized in the *quantization model* of neural scaling laws (Michaud et al., 2023), a minimal framework for deriving LM scaling laws. We encapsulate the quantization model via the following basic assumptions:

1. Language modeling consists of a large number of discrete tasks (originally called “quanta”), each of which is either unlearned or learned.
2. The distribution of tasks in the data follows a power law, resembling the Zipfian distribution of word types in natural language. Each token leverages exactly one task, and the probability of a token (in the training or test data) leveraging task k is $p_k \propto k^{-(\alpha+1)}$ for $\alpha > 0$.
3. During training, task k becomes learnable once a critical threshold T_k of relevant tokens leveraging the task have been observed. An LM learner acquires learnable tasks incrementally, spending parameters C_k on each one, in order of their rank k (formalized in Definition 4 in Section E). Once an LM learns a task, it predicts tokens leveraging that task with lower loss than before.

Under these assumptions, the loss \tilde{L} depends on the number of tasks learned, which is controlled by the budget for parameters N or training tokens D . Michaud et al. (2023) show that loss will follow a smooth power law as a function of tasks, parameters, or tokens, resulting from learning many discrete tasks one by one, with each successive task having smoothly decaying probability in the data.

Incorporating Expressivity. We now adapt the quantization model to study how the expressive power of an LM impacts scaling efficiency. Section 3 analyzed which computational tasks can be expressed by hybrid models and transformers; now we consider the *multi-task* setting, where achieving low loss requires learning many individual tasks. We first stipulate that each of these tasks is expressible or inexpressible by our LM, independent of its frequency in the data:

Assumption 1: Only Some Tasks Are Expressible

For a given architecture, each task is either expressible or inexpressible, modeled as an *iid* boolean random variable where the probability that any individual task is expressible is $1 - \epsilon$.

Increasing expressive power (e.g., going from a transformer to a hybrid model) corresponds to decreasing ϵ . We will analyze the *expected loss* under Assumption 1. Conceptually, increasing a model’s expressivity ($1 - \epsilon$) could improve scaling efficiency because it increases the proportion of tasks that a model can learn efficiently, which is the mechanism by which loss decreases. There are two plausible mechanisms: first, the model might fail to learn inexpressible tasks entirely, or, second, the model might approximate them but require more parameters and data because the architecture does not admit a compact subnetwork for solving the task. We formalize a unified *expressivity-aware* extension to the quantization model that allows for either (or both) of these effects of expressivity. First, we formalize the fact that the loss achieved on a task after it has been learned may depend on whether the task is expressible:

Assumption 2: Expressible Tasks Can Be Learned to Lower Loss

Tokens leveraging unlearned tasks incur loss L_0 . If a task k has been learned, the loss incurred by the learner on a token leveraging k is reduced to $L_0 - \Delta$ if k is expressible by that learner and to $L_0 - \Delta'$ if k is inexpressible, for some $\Delta, \Delta' > 0$ and $\Delta' \leq \Delta$.

We will refer to Δ and Δ' as *loss reductions* for expressible and inexpressible tasks, respectively. When $\Delta' = 0$, inexpressible tasks cannot be learned at all (as briefly explored by Michaud, 2026), and when $\Delta' = \Delta$, they can be fully learned, though they might require more resources (parameters and data) to learn.

We now formalize the resource requirements of different tasks. In the original quantization model (Michaud et al., 2023), all tasks require the same number of parameters C to represent and have the same sample complexity T . In contrast, we now imagine that the number of parameters and tokens needed per task can change based on whether the task is expressible:

Assumption 3: Expressible Tasks Can Be Learned with Fewer Parameters and Tokens

Each expressible task is representable with C parameters and learnable with T relevant tokens. An inexpressible task requires $C' \geq C$ parameters to represent approximately and is learnable with $T' \geq T$ relevant tokens.

Assumption 3 can be motivated as follows. If a task is fully expressible, a *small* subnetwork exists that solves the task across all inputs, requiring a reasonable number of samples to learn. If the task is inexpressible, no such small subnetwork exists, but the model can still learn a *large* lookup table that approximates the function over common inputs, requiring more samples to cover every case. This connection between expressiveness and succinctness of representation recalls philosophical arguments in theoretical computer science for analyzing computation in terms of infinite formal languages (Savitch, 1993). It is natural to imagine that expressibility leads to an exponential reduction in required resources (i.e., $C' = 2^C$), though we do not need to commit to this. The case where $C' = C$ recovers the case where inexpressible tasks do not require more parameters.

Having extended the quantization model to be expressivity-aware, we now characterize how expressivity affects scaling laws under these assumptions. The following general result establishes that increasing expressiveness leads to more parameter- and data-efficient scaling in the quantization model, and can also improve the irreducible loss achievable with infinite parameters and data.

Let $\tilde{L}(N)$ and $\tilde{L}(D)$ denote the actual loss incurred under the quantization model as a function of the number of parameters N and the token budget D .

Theorem 4: Expressivity-Aware Scaling Laws

Consider the quantization model of neural scaling laws augmented by Assumptions 1 to 3. Then, as a function of the number of parameters N , the loss $\tilde{L}(N)$ is closely approximated by a power law $L(N)$, i.e., $\tilde{L}(N) \approx L(N)$ where:

$$L(N) - L_\infty^\epsilon \propto A_\epsilon \cdot N^{-\alpha}, \quad \text{where } A_\epsilon = (L_0 - L_\infty^\epsilon) \cdot (C + \epsilon(C' - C))^\alpha.$$

Similarly, as a function of the token budget D , the loss $\tilde{L}(D)$ is closely approximated by a power law $L(D)$, i.e., $\tilde{L}(D) \approx L(D)$ where

$$L(D) - L_\infty^\epsilon \propto B_\epsilon \cdot D^{-\alpha/(\alpha+1)}, \quad \text{where } B_\epsilon = (1 - \epsilon)\Delta T^{\alpha/(\alpha+1)} + \epsilon\Delta' T'^{\alpha/(\alpha+1)}.$$

Finally, the irreducible loss L_∞^ϵ for both power laws depends on expressivity via

$$L_\infty^\epsilon = L_0 - (1 - \epsilon)\Delta - \epsilon\Delta'.$$

See Section E for a proof. Theorem 4 can be interpreted as follows. If we can express all tasks ($\epsilon = 0$), we recover the standard irreducible loss $L_\infty^0 = L_0 - \Delta$ and scaling coefficients $A_0 = \Delta C^\alpha$ and $B_0 = \Delta T^{\alpha/(\alpha+1)}$ from the quantization model (Michaud et al., 2023). On the other hand, if there are some tasks we cannot express ($\epsilon > 0$), the scaling coefficients A_ϵ and B_ϵ (and potentially irreducible loss L_∞^ϵ) will change. In particular, as illustrated in Figure 11, decreasing ϵ improves loss towards the $\epsilon = 0$ case. Formally, under any non-trivial instantiation of Assumptions 1 to 3, increasing expressivity shifts the loss curve down:

Corollary 4.1: Expressivity Always Improves Scaling

Fix a nontrivial instantiation of Assumptions 1 to 3, i.e., where either $\Delta' < \Delta$, or $C' > C$ and $T' > T$. Then, both $L(N)$ and $L(D)$ strictly decrease as ϵ decreases, elementwise for all N, D .

See justification in Section E.4. Corollary 4.1 predicts that increased expressive power will translate into more efficient model scaling. Conceptually, we can understand two separate beneficial effects, coming from Assumptions 2 and 3, respectively. First, if we instantiate Assumption 2 such that inexpressible tasks achieve a strictly lower loss reduction ($\Delta' < \Delta$), the *irreducible loss* L_∞^ϵ part of the scaling laws is affected:

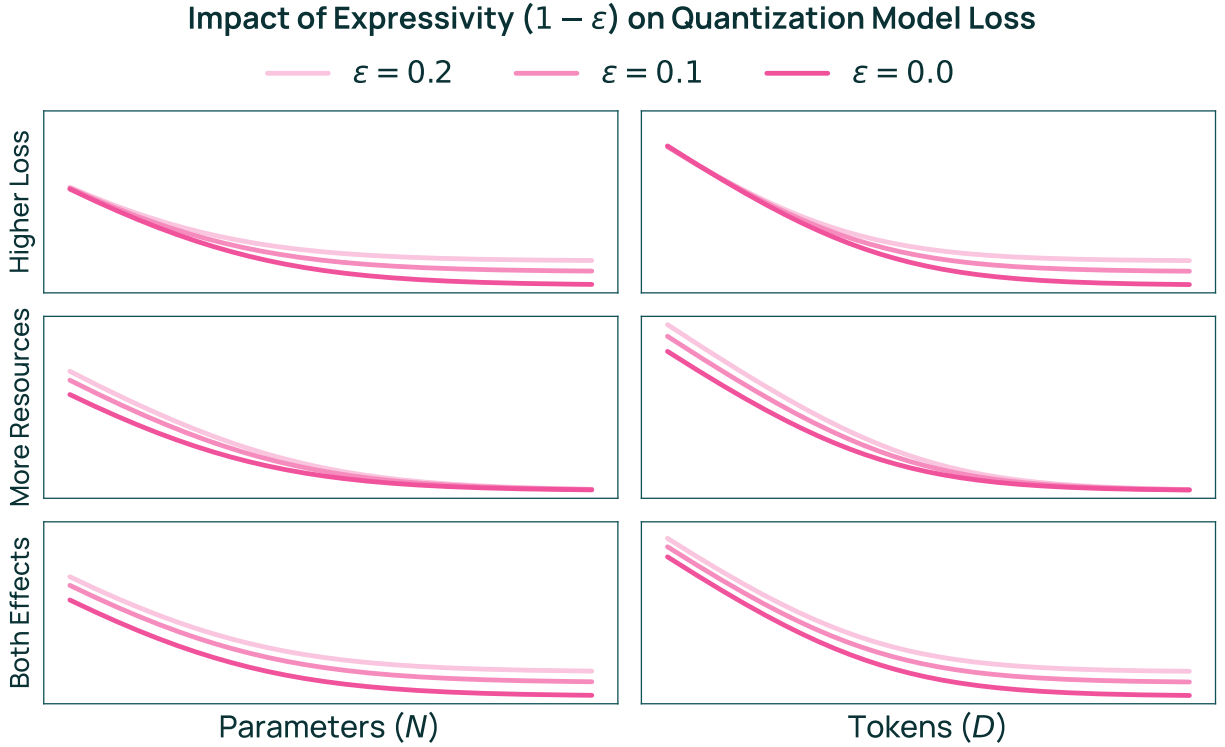


Figure 11 Impact of expressivity $1 - \epsilon$ on loss under three instantiations of the expressivity-aware quantization model; both axes are log-scaled. In the first row, $\Delta' < \Delta$, so decreasing ϵ lowers irreducible loss (Corollary 4.2) and reduces loss everywhere (Corollary 4.1), but especially for large N and D . In the second row, $\Delta' = \Delta$, but inexpressible tasks require more parameters and tokens than expressible tasks. Thus, while increasing expressivity shifts down the scaling curve everywhere, irreducible loss remains the same, so for large enough N, D , the loss reduction diminishes. Finally, the third row incorporates both effects of expressivity. This means decreasing ϵ both causes an initial difference in loss and lowers irreducible loss, leading to a visible gap in loss as ϵ decreases across values of N and D .

Corollary 4.2: Expressivity Can Shift Irreducible Loss

If and only if $\Delta' < \Delta$, irreducible loss L_∞^ϵ strictly decreases as ϵ decreases.

As visualized in the top row of Figure 11, this leads to a clear difference in loss for large N and D (when the irreducible loss starts to dominate), even though the loss curves start more similarly for small N and D . In contrast, if Assumptions 2 and 3 are instantiated so that inexpressible tasks can be fully learned ($\Delta' = \Delta$) but require more parameters and tokens ($C' > C, T' > T$), the picture is different: we see dramatic differences in loss for small values of N and D based on ϵ , but these differences go to 0 in the limit of large N and D , as illustrated in the middle row of Figure 11. Finally, if we combine both assumptions, i.e., inexpressible tasks achieve a lower loss reduction ($\Delta' < \Delta$) and also require more parameters and tokens to learn, we see a gap between the loss curves for all values of N and D , as seen in the bottom row of Figure 11.

A natural question is which instantiation of Assumptions 1 to 3 is most consistent with the empirical scaling laws from Section 4.1. Based on Corollary 4.2, lower irreducible loss for more expressive architectures supports a model where expressible tasks have greater loss reduction, i.e., $\Delta' < \Delta$. Because we do not find clear empirical evidence in Section 4.1 that architecture affects irreducible loss, it appears that a quantization model with $\Delta' = \Delta$ may better fit the data, though more precise estimates of the scaling law parameters could change this conclusion. Further, based on Corollary 4.1, a model where expressivity does not improve parameter efficiency requires $C' = C$ and $\Delta' = \Delta$. Thus, a quantization model with $T' > T$ but $C' = C$ and $\Delta' = \Delta$ appears most consistent with the observed scaling behavior of hybrid models vs. transformers, though future work that more precisely estimates the scaling coefficients (in particular, the irreducible loss) for these

architectures could change these conclusions.

4.3 Discussion: From Expressivity to Scaling

Through controlled scaling studies across model sizes and data budgets, we showed our hybrid architecture can attain better data and parameter efficiency compared to the transformer baseline. This matches the improved pretraining efficiency (both in terms of tokens and compute spent) on Common Crawl, MMLU, and other evaluations that we observed for the Olmo Hybrid pretraining run compared to Olmo 3. In Section 4.2, we argued that the pretraining efficiency of hybrid models could be explained by their greater expressivity relative to transformers, as standard explanations of scaling laws can be minimally extended to predict that more expressive models should have better loss curves.

In particular, Theorem 4 provides a conceptual explanation for why more expressive architectures might scale better. In line with our empirical findings (Section 4.1), greater expressivity improves loss across the scaling curve under our formal model but does not affect the scaling law exponent, which depend only on α , which parameterizes the task power law underlying the data distribution. Nevertheless, expressivity reduces loss across the loss curve, in line with the comparison of hybrid models and transformers in Section 4.1. As mentioned earlier, some instantiations of the quantization model (cf. Assumptions 2 and 3) also predict improved irreducible loss and parameter efficiency for more expressive models, which we do not find clear evidence for in Section 4.1.

Interestingly, and perhaps counterintuitively, even tasks that were already expressible by an architecture can be learned faster when expressivity is increased. Under Assumption 3, learning inexpressible tasks more efficiently frees up parameters than can be allocated to learning (potentially already expressible) tasks faster. This mirrors results reported by Hu et al. (2025), where allowing transformers to learn syntax more efficiently seemed to enable more efficient learning of other aspects of language.

We emphasize that our theoretical results were obtained in the simplified quantization model of scaling laws (Michaud et al., 2023), rather than by analyzing the actual learning dynamics of LMs. While this setting is somewhat simplistic, it nicely captures fundamental properties of language modeling such as its multi-task data distribution. We thus take it to provide a plausible conceptual explanation for the link between expressivity and scaling improvements, though we caution against reading too much into the precise quantitative predictions such as the scaling law coefficients and exponents (cf. Michaud, 2026). There is ample opportunity for more in-depth theoretical work to expand the analysis presented here to more realistic models of training, as well as empirical work that tests predictions of different theory variants.

While expressivity is one clear advantage of GDN-based hybrid models, other factors may also contribute to the performance gains, such as increased training stability, which we explore in Section A.1. In Section 5.1, we see that GDN hybrid models shows better scaling trends than hybrid models using Mamba, whose expressivity is constrained to TC^0 , in line with the hypothesized link between expressivity and scaling. However, GDN without the negative eigenvalue extension (which is thought to be less expressive; cf. Section 3.2) shows very similar scaling trends to GDN with negative values, in potential disagreement with theory. This could suggest that some other benefit of GDN beyond expressivity explains its improved scaling relative to transformers, though it is also an open question whether GDN, even without negative eigenvalues, could have expressivity advantages over transformers such as the ability to solve some NC^1 -complete problems.

5 Other Research Questions

5.1 RNN and Hybridization Architecture Choices

To decide on the final hybrid architecture for Olmo Hybrid, we ran a series of ablation experiments evaluating the performance of different sequence mixers and hybridization strategies. In particular, we were interested in three key design questions:

1. **RNN Architecture:** Which popular linear RNN architecture (GDN vs. Mamba2) results in the best hybrid model?
2. **Layer Placement:** How should hybridization be performed—should attention layers be interleaved

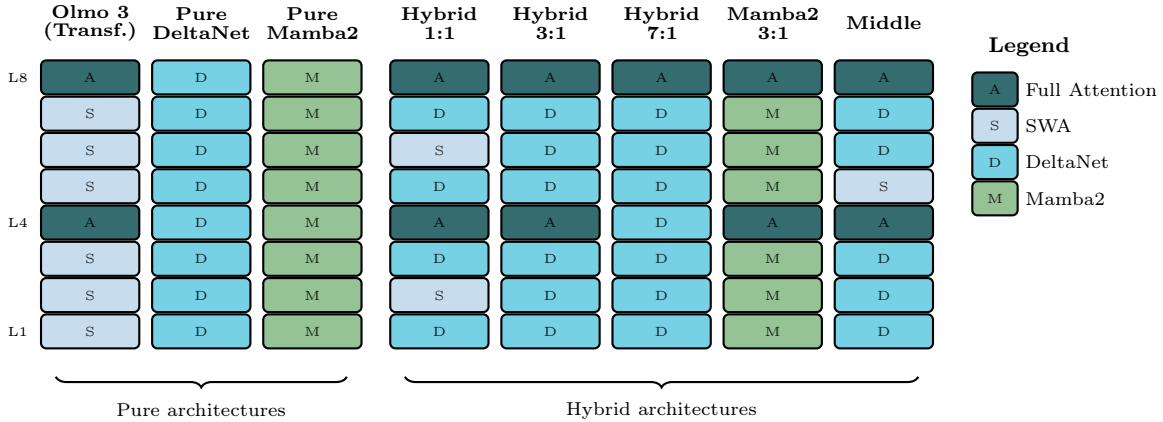


Figure 12 Architecture configurations evaluated in our ablation study. We compare pure architectures (Transformer, GDN, Mamba2) against hybrid variants with different linear-to-attention ratios (1:1, 3:1, 7:1), RNN backbones (GDN vs. Mamba2 at 3:1), and placement strategies (interleaved vs. middle). The highlighted configuration (3:1 interleaved with GDN) was selected for the final Olmo Hybrid.

uniformly throughout the model, or concentrated in specific regions?³

3. Attention Ratio: What is the optimal ratio of attention-to-RNN layers?

To inform the final model choice and provide guidelines for future work on hybrid models, we evaluated a comprehensive suite of pure and hybrid architectures at multiple scales, illustrated in Figure 12.

Experimental Setup. All ablation experiments follow the same setup as Section 4.1 and use identical training data, optimizer settings, and evaluation protocols, varying only the architecture. We train models at the same seven scales: 60M, 100M, 190M, 370M, 600M, 760M, and 1B parameters, and each model is trained on $8 \times$ Chinchilla-optimal tokens using a WSD-S learning rate schedule. Table 22 lists the detailed configurations for each model size. All hybrid models were implemented with the Flash Linear Attention library based on the Olmo 3 configuration—the only difference is the implementation of the sequence mixing component.⁴

As in Section 4.1, we measure performance as the average validation loss, computed as the average cross-entropy across 11 held-out evaluation domains: C4, Dolma Books, Dolma Common Crawl, Dolma pes2o, Dolma Reddit, Dolma Stack, Dolma Wiki, ICE, M2D2 S2ORC, Pile, and WikiText-103. Additionally, we evaluate the models on the OLMOBASEVAL Easy suite in bits per byte (BPB), reporting the average score across Math, Code, and general reasoning tasks. We again compute scaling law coefficients; in Figure 16, the curves represent the projected loss at each FLOP value at the optimal parameter and token counts, computed by fitting scaling laws analogously to Section 4.1. See also Section D.2 for more details.

Figure 16 visually shows the performance of the tested architectures and Table 5 presents the average performance on the OLMOBASEVAL Easy suite. The rest of the section interprets these results to justify the final Olmo Hybrid architecture.

RNN Architecture: GDN vs. Mamba2. We compare the two linear RNN architectures in both pure and hybrid (3:1 interleaved) settings against the transformer baseline. Pure Mamba2 performed worse than both the transformer and GDN at most scales—particularly larger ones—with average BPB of 0.72 for Mamba2 at the 1B scale compared to 0.68 for the transformer (Table 5). The hybrid Mamba2 configuration (Mamba2 + 3:1 Attn) improved over pure Mamba2 and outperformed the transformer at most scales, but fell behind at 1B (0.698 vs. 0.682). In contrast, pure GDN outperformed the transformer at all scales (e.g., 0.722 vs.

³We only consider inter-layer hybridization and did not test intra-layer hybridization strategies such as those explored in Ren et al. (2025).

⁴Thus, the position-wise MLPs were identical between the architectures. This aligns well with the GDN architecture but is slightly non-standard for Mamba2, which, as a Gated Linear Unit, contains non-linear activations in the gating mechanism. However, we found Mamba2 models without additional MLPs to perform worse, which is why we included them in the implementations used in this section.

0.747 at 760M; 0.677 vs. 0.682 at 1B), and the hybrid GDN 3:1 configuration improved further, achieving the best overall results (0.717 at 760M; 0.669 at 1B). While hybrid Mamba2 is competitive with the transformer, GDN-based models consistently outperform it across Math, Code, and QA domains (Table 21), confirming GDN as the right choice for the linear component of Olmo Hybrid.

Layer Placement: Interleaved vs. Middle. We compare two hybridization strategies that use the same 3:1 linear-to-attention ratio but place the attention layers differently:

- **Interleaved:** Attention layers placed at regular intervals (every 4th layer).
- **Middle:** Attention layers concentrated in the middle of the network, with an additional attention layer at the final position.

Despite having very similar parameter counts (948M vs. 932M at 760M; Table 22), the interleaved configuration consistently outperforms the middle placement, with the gap growing at larger scales. Both configurations outperform the transformer baseline, confirming that the benefit of hybridization is robust to placement strategy, though interleaving appears preferable. One interpretation of the advantage of interleaved placement is that uniformly distributing attention layers allows every part of the network to access global context, whereas concentrating attention in the middle creates a bottleneck through which all “attention-relevant information” must pass. Additionally, connecting to our theoretical results in Section 3.3, interleaving maximizes the number of alternations between layer types, which may unlock additional expressive power: for example, *stacked* compositions of tasks like state-based recall (Theorem 1) could benefit from multiple rounds of alternation between state tracking and recall. While Theorem 3 shows that a single alternation suffices with padding tokens, in practice, multiple alternations without padding may be more effective.

Linear-to-Attention Ratio: 1:1 vs. 3:1 vs. 7:1. We vary the fraction of attention layers from 50% down to 12.5%, using interleaved placement with GDN throughout. At the smallest scales (60M–190M parameters), the 7:1 ratio (12.5% attention) tends to perform best or on par with 3:1, suggesting that fewer attention layers can suffice when models are small. At larger scales (600M–1B), however, the 3:1 ratio consistently achieves the best or second-best performance across all domains (Table 21), while the 7:1 ratio falls slightly behind despite having more parameters. The 1:1 ratio (50% attention) performs comparably to 3:1 at small scales but underperforms at larger scales, suggesting that the higher computational cost of more attention layers does not pay off relative to the GDN layers. Overall, the 3:1 ratio (25% attention) offers the best trade-off across scales and domains, and we select it for Olmo Hybrid. Notably, all three interleaved hybrid configurations substantially outperform the transformer at large scales, so the exact ratio is less critical than the decision to hybridize at all.

GDN Architecture: Gate and Eigenvalue Sign. We additionally ablate two internal GDN design choices—the use of the output gate and the sign of the recurrence eigenvalues—across both pure and hybrid (3:1 interleaved) configurations. In the GDN, the *output gate* multiplies the RNN output by a learned sigmoid-gated projection, while the *eigenvalue sign* determines whether the recurrence allows oscillatory dynamics (negative EVs) or monotone decay only (positive EVs). Results are reported in the bottom two sections of Tables 5 and 21.

For **pure GDN**, removing the gate consistently hurts performance across all scales and domains: both no-gate variants (Neg EV, no gate and Pos EV, no gate) underperform their gated counterparts, confirming the gate is a useful component in a pure-RNN setting. Interestingly, positive EVs with a gate (Pos EV, gate) perform comparably to or slightly better than negative EVs with a gate at several scales, though the differences are small and inconsistent.

For **hybrid GDN (3:1)**, the picture changes: the selected architecture (Neg EV, gate) remains the most consistent performer, but the no-gate variants are more competitive in the hybrid setting than in the pure setting, and occasionally outperform the gated variants at individual scales. This suggests that, in the hybrid setting, attention layers may partially compensate for the removed gate, reducing its marginal value. Overall, the differences among the four hybrid variants are small (within ~ 0.01 BPB at most scales), and we retain the gated negative-EV configuration as the default based on its slight edge in consistency.

Table 5 Architecture ablation results – averages. Average OLMOBASEVAL BPB (across Math, Code, QA) for pure and hybrid architectures at 7 representative scales. **Bold** indicates best; underline second best; † best within section. ★ marks our selected architecture. Note that per-size comparisons should be interpreted with care, as architectures at the same nominal scale differ in actual parameter count (see Table 22).

Architecture	Attn %	60M	100M	190M	370M	600M	760M	1B
<i>Pure Architectures</i>								
Transformer	100%	1.155	1.037	0.950	0.839	0.781	0.747	0.682
GDN	0%	1.080 [†]	0.990 [†]	0.895 [†]	0.795	0.761 [†]	0.722 [†]	0.677 [†]
Mamba2	0%	1.146	1.036	0.941	0.843	0.787	0.750	0.718
<i>Hybrid: Interleaved Attention</i>								
GDN (1:1)	50%	1.093	0.992	0.896	0.804	0.787	0.724	0.674
GDN (3:1)★	25%	<u>1.077</u>	0.986 [†]	0.891	0.799 [†]	0.754	<u>0.717</u>	0.669 [†]
GDN (7:1)	12.5%	1.082	0.988	0.892	0.803	0.748	0.721	0.675
Mamba2 (3:1)	25%	1.130	1.012	0.921	0.828	0.772	0.732	0.698
<i>Hybrid GDN (3:1): Middle Placement</i>								
Interleaved★	25%	1.077 [†]	0.986 [†]	<u>0.891</u>	0.799 [†]	0.754	0.717 [†]	0.669 [†]
Middle	25%	1.087	0.988	0.899	0.800	0.754 [†]	0.721	0.672
<i>Hybrid GDN (3:1): Gate/Eigenvalue Ablations</i>								
Neg EV, gate★	25%	1.077 [†]	0.986	0.891 [†]	0.799	0.754	0.717	0.669
Neg EV, no gate	25%	1.081	0.976	0.894	0.801	0.752 [†]	0.721	0.666
Pos EV, gate	25%	1.086	<u>0.981</u>	0.897	0.828	0.757	0.714	<u>0.667</u>
Pos EV, no gate	25%	1.082	0.988	0.901	0.795 [†]	0.754	0.717	0.670
<i>Pure GDN: Gate/Eigenvalue Ablations</i>								
Neg EV, gate	0%	1.080	0.990	0.895	<u>0.795</u>	0.761	0.722	0.677
Pos EV, gate	0%	1.070	0.982 [†]	0.893 [†]	0.798	<u>0.751</u>	0.720 [†]	0.674 [†]
Neg EV, no gate	0%	1.087	0.999	0.904	0.809	0.761	0.731	0.680
Pos EV, no gate	0%	1.092	1.002	0.901	0.806	0.765	0.727	0.678

Summary of Findings. The ablation study allows us to answer the three design questions posed at the beginning of this section. First, **GDN is preferred over Mamba2** as the linear RNN component: the Mamba2-based models performed worse at every scale in both pure and hybrid configurations, while GDN matched or outperformed the transformer even as a pure architecture. Second, **interleaved placement outperforms middle placement**: distributing attention layers uniformly throughout the network consistently yields better results than concentrating them, likely because it allows every part of the network to access global context and maximizes the number of layer-type alternations. Third, **the 3:1 linear-to-attention ratio is a good default overall choice**: while the 7:1 ratio is competitive at small scales, the 3:1 ratio provides the most consistent gains at large scales, and the 1:1 ratio does not justify the additional attention cost.

5.2 Olmo Hybrid vs. Other Open Models

Setup. We organize our baseline open-weight models into four groups, distinguishing dense vs. MoE MLP layers and RNN-only vs. Hybrid vs. Attention-only architectures. We emphasize that baselines were trained with different datasets and token budgets, making direct comparisons between them less meaningful for evaluating architecture choices.

The closest group to Olmo Hybrid are other hybrid models with dense MLP layers: Nemotron-H (NVIDIA, 2025), Falcon H1 (Zuo et al., 2025), and RecurrentGemma (Botev et al., 2024). Falcon H1 is a parallel hybrid mixer (also referred to as “intra-layer hybridization”): it uses Mamba-2 SSM and attention blocks on 100% of layers and performs a channel-wise concatenation prior to the MLP layer. Nemotron-H is the closest architecture to ours, with 8% of layers using self-attention (GQA; Ainslie et al., 2023) and all other layers

using Mamba-2 SSM blocks. RecurrentGemma applies local self-attention (with a 2K context window) and an SSM using the Griffin architecture (De et al., 2024) on 100% of layers and takes a gated sum.

We also compare to open-weight pure RNN models: Falcon Mamba (Zuo et al., 2024) and xLSTM (Beck et al., 2025). Falcon Mamba uses the Mamba 1 architecture on 100% of layers. xLSTM uses mLSTM (Beck et al., 2026) (a gated linear RNN with a memory state) on 100% of layers, and is the only model running in FP32.

Finally, we report performance for hybrid models using MoE for MLP layers: Nemotron 3 Nano (NVIDIA Team, 2025b), which alternates between Mamba-2 SSM on 85% of layers and self-attention (GQA) on 15% of layers; and Kimi Linear (Kimi Team, 2025), which uses Kimi DeltaNet on 75% of layers and self-attention (MLA; DeepSeek-AI, 2024) on 25% of layers.

Evaluation Details. Not all models apply long-context extension to the same window size, so for our base model results on RULER we evaluate up to the maximum supported context window at the end of pretraining. We use the most recent vLLM (Kwon et al., 2023) and transformers versions across all models, except for the pure RNN baselines⁵.

Findings. As shown in Table 6, we report base model performance on OLMOBASEEVAL, including reported parameter and token counts. For each baseline, we estimate training compute using the FLOPs = $6ND$ heuristic from Kaplan et al. (2020) and FLOPs = $6N_{\text{active}}D$ for MoE models following Fedus et al. (2022); Clark et al. (2022). While all dense baselines have a similar number of parameters (7–9B), they were trained across a wide range of token budgets, presenting a clear compute-performance tradeoff.

Olmo Hybrid substantially outperforms both pure RNN baselines, including Falcon Mamba (which is closely matched on tokens and parameters), across all task averages in OLMOBASEEVAL. Olmo Hybrid also outperforms the older open-weight model xLSTM, trained on roughly 2T tokens.

Across OLMOBASEEVAL task averages, Olmo Hybrid 7B shows competitive performance against both Nemotron-H 8B (trained for 15T tokens, $2.5\times$ more than Olmo Hybrid) and Falcon H1 (trained for 12T tokens, $2\times$ as many as Olmo Hybrid).

Among hybrid dense models, there is a large discrepancy in token budgets. Olmo Hybrid outperforms RecurrentGemma, which was trained on $3\times$ fewer tokens. It also appears generally strong for the amount of data it was trained on, matching or outperforming Nemotron-H and Falcon H1 (which have $2\times$ larger token budgets) on some tasks, while performing worse on others (e.g., Olmo Hybrid only outperforms Falcon H1 on GenQA tasks).

5.3 Post-Training Olmo Hybrid

As a preliminary investigation of hybrid models’ capabilities after post-training, we apply a portion of the Olmo 3 post-training pipeline to create a preliminary Instruct version of Olmo Hybrid. Following Olmo Team (2025), this involves three stages: thinking SFT on long reasoning traces, then an instruction-tuning phase *without* the model first thinking, and finally direct preference optimization (DPO; Rafailov et al., 2024). In this section, we compare Olmo Hybrid and Olmo 3 7B at each stage and discuss details and challenges we encountered when adapting our post-training recipe to the hybrid architecture.

Data Changes. Relative to Olmo 3, there is one minor change to the Think SFT stage: the addition of function-calling data for tool use. The lack of advanced tool use in our Think models is a known limitation for building agentic models; this new data is one part of broader changes planned for future Olmo models. We use the same prompts from the Olmo 3 Instruct SFT models with new thinking traces, reusing thinking traces from DR Tülu (Shao et al., 2025) for the Web Search QA prompts and generating reasoning traces with GPT-4.1 for the remaining data, upsampling each data point $3\times$ to increase token coverage.⁶ The Instruct SFT and DPO data are identical to the Olmo 3 7B and 32B Instruct SFT and DPO variants.⁷

⁵We observed slight differences in scores from those reported in the Olmo 3 paper ($<0.25\%$ absolute difference for all tasks for Olmo 3) when using the most recent `transformers==5.0.0`. As `transformers==5.0.0` breaks both pure RNN baselines (RecurrentGemma and xLSTM), we use older versions for those baselines.

⁶The new Think SFT dataset is available at: <https://hf.co/datasets/allenai/Dolci-Think-SFT-Olmo-Hybrid>

⁷SFT: <https://hf.co/datasets/allenai/Dolci-Instruct-SFT>. DPO: <https://hf.co/datasets/allenai/Dolci-Instruct-DPO>

	Hybrid Dense				Pure RNN Dense		Hybrid MoE		Dense Baselines	
	Olmo Hybrid	Nemo.-H	Falcon H1	Recurr. Gemma	Falcon Mamba	xLSTM	Nemo. 3 Nano	Kimi Linear	Olmo 3	Qwen 3
# Parameters	7B	8B	7B	9B	7B	7B	30B A3B	48B A3B	7B	8B
# Train Tokens	6T	15T	12T	2T	6T	2T	25T	6T	6T	36T
Train Compute (10 ²³ FLOPs)	2.6	7.2	5.5	1.0	2.5	0.9	5.4	1.0	2.6	17.3
OlmoBaseEval Math	55.1	54.6	65.7	32.1	33.7	18.3	53.2	68.5	54.6	67.2
GSM8k	74.3	76.8	80.9	49.6	57.1	32.8	86.3	84.9	75.2	84.2
GSM Symbolic	49.2	55.4	64.8	25.3	25.8	10.8	68.6	66.7	48.4	65.4
MATH	41.8	31.7	51.4	21.3	18.2	11.3	4.6	54.0	40.1	52.0
OlmoBaseEval Code	32.4	37.1	45.3	23.7	14.6	3.1	47.3	30.3	30.9	46.2
BigCodeBench	35.1	40.3	40.5	20.2	0.2	4.0	45.7	44.5	34.8	43.1
HumanEval	49.0	60.1	61.0	34.5	0.1	13.3	77.1	72.4	49.0	71.2
DeepSeek LeetCode	2.2	4.1	3.5	0.6	0.1	0.0	10.1	1.1	1.5	8.8
DS 1000	21.1	24.7	29.0	19.1	16.3	2.3	33.3	34.3	20.6	33.3
MBPP	50.3	58.0	63.5	33.7	37.5	1.3	66.4	58.3	43.6	65.8
MultiPL HumanEval	29.4	32.3	59.7	21.6	15.7	0.2	48.3	1.1	28.8	52.5
MultiPL MBPPP	39.5	40.2	59.8	36.4	32.1	0.3	50.0	0.6	38.3	48.5
OlmoBaseEval MC STEM	70.0	72.4	75.7	61.6	64.2	36.9	78.6	77.5	66.2	78.7
ARC MC	90.8	93.3	94.2	82.6	85.7	46.7	94.8	94.3	89.2	95.4
MMLU STEM	64.6	64.2	73.8	49.6	52.0	34.3	73.8	67.8	59.7	76.7
MedMCQA MC	52.1	56.2	59.9	47.3	48.9	31.4	64.2	64.6	48.0	63.5
MedQA MC	48.7	54.5	55.9	39.4	42.5	23.2	64.9	66.6	41.6	62.0
SciQ MC	93.9	94.1	94.7	88.9	91.9	48.8	95.0	94.0	92.8	96.0
OlmoBaseEval MC Non-STEM	80.4	80.7	84.1	71.1	74.2	39.9	83.9	76.2	78.2	84.9
MMLU Humanities	71.6	76.8	78.7	62.1	65.8	37.6	80.5	79.0	69.2	78.6
MMLU Social Sci.	79.7	80.5	84.4	68.4	70.8	39.5	85.1	73.4	75.2	84.9
MMLU Other	71.0	73.2	76.3	63.9	64.1	38.7	78.0	78.2	66.8	76.7
CSQA MC	78.4	75.9	78.1	67.8	73.0	32.5	75.3	52.8	75.2	84.1
PiQA MC	82.7	85.9	87.8	76.4	83.5	51.5	90.0	82.8	80.2	89.9
SocialIQA MC	81.1	78.6	81.5	75.5	77.5	34.1	80.7	78.8	80.3	83.3
CoQA Gen2MC MC	93.9	91.6	94.4	83.3	86.5	33.6	92.2	91.2	92.5	93.6
DROP Gen2MC MC	68.1	60.9	81.3	48.7	51.4	33.9	70.0	71.9	67.4	78.4
Jeopardy Gen2MC MC	89.3	92.5	91.8	83.9	90.2	46.7	94.9	54.1	86.9	92.1
NaturalQs Gen2MC MC	71.0	76.0	73.3	64.2	67.7	34.3	79.1	78.1	69.5	74.6
SQuAD Gen2MC MC	97.0	95.8	97.5	87.6	85.4	56.2	97.3	97.3	97.0	97.4
OlmoBaseEval GenQA	72.9	71.2	71.7	68.5	68.5	34.8	78.1	75.7	72.5	71.1
HellaSwag RC	79.0	84.3	80.9	81.3	81.9	61.1	86.0	85.0	77.7	80.6
Winogrande RC	86.2	89.6	87.9	86.3	88.3	62.1	88.7	88.2	85.6	86.4
Lambada	70.2	76.0	73.0	72.2	80.9	19.7	75.9	77.2	68.8	72.8
Basic Skills	89.7	90.2	93.3	83.0	86.1	60.9	92.5	93.1	89.5	93.4
DROP	72.9	65.7	69.4	44.0	41.6	24.4	75.0	72.2	71.5	57.2
Jeopardy	64.0	69.7	61.1	66.4	67.1	9.3	77.2	77.1	60.3	65.1
NaturalQs	34.8	37.7	27.4	30.6	35.1	1.8	45.7	25.9	32.5	33.8
SQuAD	92.1	87.9	91.4	85.9	75.9	33.9	94.0	91.8	93.6	89.0
CoQA	67.4	39.9	60.7	66.6	59.7	40.4	67.7	70.3	72.8	61.8
OlmoBaseEval HeldOut										
LBPP	16.8	26.8	30.2	5.8	5.7	0.6	33.7	31.1	17.7	26.2
BBH	65.2	69.9	75.5	53.3	42.8	24.6	78.2	68.6	64.0	76.5
MMLU Pro MC	41.7	44.4	50.0	27.6	24.5	11.4	53.5	50.7	37.2	50.1
Deepmind Math	23.4	26.1	34.3	17.3	15.5	7.5	32.8	40.8	23.6	47.6
Long-context Eval										
RULER 4K	92.7	84.1	92.0	44.9	68.2	28.6	96.2	88.5	94.9	95.5
RULER 8K	91.4	82.0	92.0	-	49.0	20.0	95.2	89.4	91.2	94.1
RULER 16K	90.0	-	89.7	-	23.8	12.4	93.8	94.0	84.1	93.5
RULER 32K	86.4	-	83.0	-	4.7	-	90.3	90.2	78.3	90.1
RULER 64K	85.0	-	77.2	-	0.0	-	85.9	90.5	67.8	-

Table 6 Base model performance on OLMOBASEEVAL compared to open-weight baselines using the OLMOBASEEVAL suite. Olmo Hybrid outperforms the dense Olmo 3 on all OLMOBASEEVAL multi-task averages, and both Pure RNN baselines (Falcon Mamba and xLSTM). Olmo Hybrid is also competitive with other open-weight hybrid dense models, like Nemotron-H and Falcon H1, despite being trained on half or fewer tokens. Olmo Hybrid was not evaluated on held-out benchmarks prior to release. RULER was evaluated up to the max context window for each model.

Decoding Settings. Finding a stable configuration for Olmo Hybrid required careful attention to vLLM flag settings and implementation details, particularly for evaluation. Correct generation and throughput were both sensitive to careful choice of these settings, although this will likely change as hybrid models are more widely adopted and open-source tooling improves. The two key flags needed to get maximum performance with the post-trained models were `--disable-cascade-attn`, which disables cascade attention (an optimization for shared prompt prefixes), and `--enforce-eager`, which disables torch compilation. These two flags have been used in our RL setup dating back to Olmo 3, but are new additions to evaluations, and scores drop precipitously without them. We hypothesize that `--enforce-eager` matters more for Olmo

Model	Knowledge & Reasoning					Math				Code			Chat & IF		
	MMLU	PopQA	BBH	GPQA	Zebra	MATH	Ω	AIME'25	AIME'24	HE+	MBPP+	LCB	IFEval	IFB	AE3
Olmo 3 Think SFT	74.9	20.8	84.1	45.8	57.9	94.4	37.8	57.6	69.6	88.2	63.2	67.8	77.9	30.0	43.9
Olmo Hybrid Think SFT	80.5	25.1	84.6	47.0	55.1	93.8	35.1	55.2	66.2	86.3	63.7	65.5	80.4	31.6	49.0
Olmo 3 Instruct SFT	67.1	16.5	51.0	30.0	18.0	65.1	14.4	7.2	6.7	69.8	56.5	20.0	81.7	27.4	21.8
Olmo Hybrid Instruct SFT	71.9	16.8	47.3	36.8	17.0	66.7	16.0	8.8	6.7	69.2	55.3	21.3	81.5	29.0	25.6
Olmo 3 DPO	69.1	20.7	69.3	37.9	28.4	79.6	22.8	20.4	23.5	72.9	55.9	18.8	82.0	29.3	43.3
Olmo Hybrid DPO	73.6	21.0	57.3	38.0	29.1	72.9	19.5	10.2	10.1	75.1	56.9	22.0	80.5	33.3	56.3
<i>Hybrid – Dense (Δ)</i>															
Δ Think SFT	+5.6	+4.3	+0.5	+1.2	-2.8	-0.6	-2.7	-2.4	-3.4	-1.9	+0.5	-2.3	+2.5	+1.6	+5.1
Δ Instruct SFT	+4.8	+0.3	-3.7	+6.8	-1.0	+1.6	+1.6	+1.6	-0.0	-0.7	-1.2	+1.3	-0.3	+1.6	+3.8
Δ DPO	+4.5	+0.3	-12.0	+0.1	+0.7	-6.7	-3.3	-10.2	-13.4	+2.2	+1.0	+3.2	-1.5	+4.0	+13.0

Table 7 Post-training evaluation comparison between Olmo 3 (dense) and Olmo Hybrid (hybrid) across three training stages: Think SFT, Instruct SFT, and DPO. Ω = Omega Full. HE+ = HumanEvalPlus. IFB = IFBench. AE3 = AlpacaEval 3.

Model	Metric	1K	4K	8K	16K	32K
Olmo 3 7B (MHA)	Tokens/sec/node	2,909	2,164	1,510	1,690	1,247
	Inference GPUs (RL batch 1K)	32	48	56	64	96
	Generation wall time (h)	10.5	56.7	162.5	283.7	768.6
	GPU-hours (\times 1K)	0.3	2.7	9.1	18.2	73.8
Olmo Hybrid 7B	Tokens/sec/node	3,039	3,077	2,736	1,876	1,422
	Inference GPUs (RL batch 1K)	8	16	24	40	64
	Generation wall time (h)	10.1	39.9	89.6	261.5	689.9
	GPU-hours (\times 1K)	0.1	0.6	2.2	10.5	44.2
Olmo Hybrid 7B (enforce eager)	Tokens/sec/node	1,410	1,565	1,592	1,495	1,066
	Inference GPUs (RL batch 1K)	8	16	24	40	64
	Generation wall time (h)	21.7	78.4	154.1	328.1	920.4
	GPU-hours (\times 1K)	0.2	1.3	3.7	13.1	58.9

Table 8 Inference-bound scaling for async RL generation across context lengths. In async RL, each training step requires generating a full batch of rollouts before the policy update; generation throughput is the bottleneck. We measure the minimum GPU allocation needed to serve an RL batch of 1,024 rollouts concurrently and project generation wall time and total GPU-hours over 1,725,440 episodes (1,684 steps).

Hybrid than for a dense transformer because torch compilation can introduce subtle numerical differences that compound across recurrent GDN state updates in a way that standard attention layers do not experience. Consistent with this interpretation, re-enabling compilation while storing the GDN cache in FP32 via `--mamba_ssm_cache_dtype float32` (NVIDIA Team, 2025a) recovers similar scores, suggesting that repeated low-precision downcasting across recurrent steps is part of the issue.

Inference Throughput. We conducted early investigations into completing our recipe with reinforcement learning with verifiable rewards (RLVR; Lambert et al., 2025). To quantify basic inference throughput—the bottleneck in our synchronous Olmo RL setup (Olmo Team, 2025)—we benchmarked Olmo Hybrid on a single $8\times A100$ node with a 2,048-token prompt, 2 degrees of tensor parallelism, and a batch size of 16 prompts, each prompt sampled $4\times$. The details are shown in Table 7. Without `--enforce-eager`, Olmo Hybrid is competitive with or faster than the dense OLMo 3 7B (4K: 3,023 vs. 2,164, 8K: 2,247 vs. 1,510, 16K: 1,486 vs. 1,690, and 32K: 1,499 vs. 1,247 tokens/sec/node), largely because the gated delta net layers reduce KV cache memory pressure compared to the full multi-head attention used in Olmo 3 7B (which lacks GQA). However, enabling `--enforce-eager` – as required for numerical stability – substantially reduces throughput (4K: 1,425, 8K: 1,365, 16K: 1,197, and 32K: 654 tokens/sec), roughly $0.5\text{--}0.9\times$ that of the dense baseline. The `--enforce-eager` flag was also used in our Olmo 3 training runs, but the dense model did not suffer from

this substantial slowdown.⁸

Post-Training Performance. Table 7 shows the performance of the Think SFT, Instruct SFT, and DPO checkpoints for Olmo Hybrid compared to Olmo 3. Overall, the stronger pretraining performance translates to persistent gains on knowledge tasks, but the model still lags behind Olmo 3 on extended reasoning tasks such as AIME and Omega (Sun et al., 2025). We anticipate that adapting the post-training data for the hybrid model could improve performance on these benchmarks. Beyond the data, early post-training results were also sensitive to decoding kernels and related settings, so improvements there could further close the gap. Other work has shown that the strongest teacher model does not always improve the downstream student proportionally (Guha et al., 2025; OpenThoughts-Agent Team, 2025), which could be base-model-specific, further highlighting the need to iterate on the post-training data beyond what was used for Olmo 3.⁹ At the level of both engineering and research, post-training hybrid models is in its infancy; we will continue to work on these recipes and share our findings with the community.

One limitation of our current investigation into post-training is that we have not considered the implications of switching to a hybrid architecture for safety. In future work, it would be interesting to assess whether Olmo Hybrid exhibits different behavior on safety evaluations compared to transformers.

6 Conclusion

Compared to Olmo 3, Olmo Hybrid achieves better pretraining efficiency, which translates to improvements on downstream tasks, including long-context abilities. While there have been many releases of strong hybrid models mixing recurrence with attention, our results provide evidence for the benefit of hybrid architectures over transformers in a controlled, large-scale setting. Beyond these headline results, we present theoretical analysis and fully controlled scaling studies that further substantiate these advantages. In particular, we show that hybrid models are more expressive than the sum of their parts: they can represent synthetic tasks that neither transformers nor RNNs in isolation can. We also explore the theoretical link between expressivity and language model scaling, showing that standard conceptual explanations for scaling laws predict that increasing expressivity should benefit training efficiency, consistent with our empirical findings. Overall, Olmo Hybrid provides evidence for hybrid models over transformers and a conceptual explanation for their observed gains. This work raises many questions for future research: post-training hybrid models, optimizing RNN architecture details, and more deeply understanding the connection between an architecture’s expressive power and its scaling behavior.

⁸Subsequent improvements to our vLLM implementation have recovered much of the throughput gap between Olmo Hybrid and OLMo 3 7B in eager mode: <https://github.com/vllm-project/vllm/pull/32550#issuecomment-3994432476>.

⁹The long-context recipe used for Olmo Hybrid also differs slightly from Olmo 3 7B, which could contribute to differences in post-training behavior.

Author Contributions

- **William Merrill** led the project, contributing to early experiments, pretraining, theory, and writing.
- **Yanhong Li** conducted early proof-of-concept experiments, implemented the HuggingFace and vLLM versions of Olmo Hybrid, ran evaluations, and owned the synthetic experiments and their writeup.
- **Tyler Romero** ran pretraining, mid-training, and long-context extension. He also optimized training throughput, implemented Ulysses-style context parallelism, improved the throughput and numerical correctness of Olmo Hybrid’s vLLM implementation, and helped with writing.
- **Anej Svete** ran early proof-of-concept experiments alongside WM, ran the experiments for the empirical scaling laws and architecture ablation sections, and wrote up those sections.
- **Caia Costello** led the collaboration from Lambda’s side, including project scoping, data management, preliminary research on Olmo Hybrid training across B200 and H100 hardware, and real-time support for pretraining.
- **Pradeep Dasigi** generated tool use data for Olmo Hybrid Think SFT and helped with artifact release logistics.
- **Dirk Groeneveld** helped plan the Olmo Hybrid pretraining run and analyze the training stability of Olmo Hybrid.
- **David Heineman** ran generative evaluations for Olmo Hybrid and open-weight models and wrote up these results.
- **Bailey Kuehl** handled technical aspects of preparing Olmo Hybrid artifacts and documentation for release.
- **Nathan Lambert** led post-training for Olmo Hybrid and wrote up post-training results.
- **Chuan Li** provided high-level supervision on the Lambda side of the project.
- **Kyle Lo** contributed to pre-training evaluations, writing, and presentation.
- **Saumya Malik** contributed to post-training methodology (tokenization and chat templates) and evaluation.
- **DJ Matusz** helped validate the compatibility of Olmo Hybrid pretraining with B200 hardware.
- **Benjamin Minixhofer** implemented FLA in OLMo-core to support the initial experiments for Olmo Hybrid and also benchmarked inference throughput.
- **Jacob Morrison** contributed to post-training, focusing on SFT.
- **Luca Soldaini** helped plan initial experiments and pretraining, and contributed to vLLM implementation and post-training debugging.
- **Finbarr Timbers** contributed to post-training Olmo Hybrid, implementing the hybrid model in open-instruct.
- **Pete Walsh** worked on training code, built the Slurm launch system for pretraining on Lambda’s cluster, and helped run pretraining.
- **Noah Smith** provided high-level guidance on many aspects of the project.
- **Hannaneh Hajishirzi** provided high-level guidance on many aspects of the project.
- **Ashish Sabharwal** provided guidance on the initial experiments and made core technical contributions to the theoretical aspects of the project.

Acknowledgments

The authors thank Taira Anderson, Kyle Wiggers, David Albright, and Stephen Kelman for contributing to the release of Olmo Hybrid. WM thanks Songlin Yang and Mehryar Mohri for relevant discussions. AS acknowledges the support of the ETH AI Center doctoral fellowship. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Additionally, this research used Lambda’s computational resources for part of Olmo Hybrid pretraining; the authors thank Amir Zadeh, Allison Beck, Abhi Sarma, and Long Fei for their support during that phase. Finally, this material is based upon work supported by the National Science Foundation under Award No. 2413244.

References

- J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL <https://arxiv.org/abs/2305.13245>.
- E. Akyürek, B. Wang, Y. Kim, and J. Andreas. In-context language learning: Architectures and algorithms. In *ICML*, 2024. URL <https://proceedings.mlr.press/v235/akyurek24a.html>.
- S. Arora and A. Goyal. A theory for emergence of complex skills in language models, 2023. URL <https://arxiv.org/abs/2307.15936>.
- S. Arora, S. Eyuboglu, A. Timalsina, I. Johnson, M. Poli, J. Zou, A. Rudra, and C. Re. Zoology: Measuring and improving recall in efficient language models. In *ICLR*, 2024a. URL <https://openreview.net/forum?id=LY3ukUANko>.
- S. Arora, S. Eyuboglu, M. Zhang, A. Timalsina, S. Alberti, J. Zou, A. Rudra, and C. Re. Simple linear attention language models balance the recall-throughput tradeoff. In *ICML*, 2024b. URL <https://openreview.net/forum?id=e93ffDcpH3>.
- J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models, 2021. URL <https://arxiv.org/abs/2108.07732>.
- D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . In *STOC*, pages 1–5, 1986. URL <https://dl.acm.org/doi/10.1145/12130.12131>.
- M. Beck, K. Pöppel, P. Lippe, R. Kurle, P. M. Blies, G. Klambauer, S. Böck, and S. Hochreiter. xLSTM 7B: A recurrent LLM for fast and efficient inference, 2025. URL <https://arxiv.org/abs/2503.13427>.
- M. Beck, K. Schweighofer, S. Böck, S. Lehner, and S. Hochreiter. xLSTM scaling laws: Competitive performance with linear time-complexity. In *ICLR*, 2026. URL <https://openreview.net/forum?id=bpbU549sSg>.
- Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. *AAAI*, 34(05):7432–7439, 2020.
- A. Botev, S. De, S. L. Smith, A. Fernando, G.-C. Muraru, R. Haroun, L. Berrada, R. Pascanu, P. G. Sessa, R. Dadashi, et al. RecurrentGemma: Moving past transformers for efficient open language models, 2024. URL <https://arxiv.org/abs/2404.07839>.
- J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. In *ICLR*, 2017. URL <https://openreview.net/forum?id=H1zJ-v5x1>.
- S. R. Buss. The boolean formula value problem is in $ALOGTIME$. In *STOC*, pages 123–131, 1987. doi: 10.1145/28395.28409. URL <https://dl.acm.org/doi/10.1145/28395.28409>.
- F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, et al. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation, 2022. URL <https://arxiv.org/abs/2208.08227>.
- H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998. doi: 10.1006/jcss.1998.1588. URL <https://doi.org/10.1006/jcss.1998.1588>.
- M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- D. Chiang. Transformers in uniform TC^0 . *TMLR*, 2025. URL <https://openreview.net/forum?id=ZA7D4nQuQF>.
- A. Clark, D. de Las Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud, et al. Unified scaling laws for routed language models. In *ICML*, pages 4057–4086, 2022. URL <https://proceedings.mlr.press/v162/clark22a.html>.
- P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018. URL <https://arxiv.org/abs/1803.05457>.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>.

- T. Dao and A. Gu. Transformers are SSMS: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024. URL <https://openreview.net/forum?id=ztn8FCGNny>.
- S. De, S. L. Smith, A. Fernando, A. Botev, G. Cristian-Muraru, A. Gu, R. Haroun, L. Berrada, Y. Chen, S. Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024. URL <https://arxiv.org/abs/2402.19427>.
- DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL <https://arxiv.org/abs/2405.04434>.
- N. S. Dey, B. C. Zhang, L. Noci, M. Li, B. Bordelon, S. Bergsma, C. Pehlevan, B. Hanin, and J. Hestness. Don't be lazy: CompleteP enables compute-efficient deep transformers. In *NeurIPS*, 2025. URL <https://openreview.net/forum?id=1MU2kaMAN1>.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*, 2019. URL <https://aclanthology.org/N19-1246>.
- Y. Dubois, B. Galambosi, P. Liang, and T. B. Hashimoto. Length-controlled AlpacaEval: A simple way to debias automatic evaluators, 2024. URL <https://arxiv.org/abs/2404.04475>.
- J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1. URL https://doi.org/10.1207/s15516709cog1402_1.
- W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(120):1–39, 2022. URL <https://jmlr.org/papers/v23/21-0998.html>.
- L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL <https://arxiv.org/abs/2101.00027>.
- Y. Gelberg, K. Eguchi, T. Akiba, and E. Cetin. Extending the context of pretrained LLMs by dropping their positional embeddings, 2025. URL <https://arxiv.org/abs/2512.12167>.
- S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992. doi: 10.1162/neco.1992.4.1.1. URL <https://doi.org/10.1162/neco.1992.4.1.1>.
- S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. In *ICLR*, 2024. URL <https://openreview.net/forum?id=ph04CRkPdC>.
- R. Grazi, J. Siems, J. K. Franke, A. Zela, F. Hutter, and M. Pontil. Unlocking state-tracking in linear RNNs through negative eigenvalues. In *ICLR*, 2025. URL <https://openreview.net/forum?id=UvTo3tVBk2>.
- S. Greenbaum and G. Nelson. The international corpus of English (ICE) project. *World Englishes*, 15(1):3–15, 1996. doi: 10.1111/j.1467-971x.1996.tb00088.x.
- R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *A Compendium of Problems Complete for P*. Citeseer, 1991. URL <https://www.semanticscholar.org/paper/A-Compendium-of-Problems-Complete-for-P-Greenlaw-Hoover/32f9d1b1bcead912c87de195a6b3695b13eb3d95>.
- A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL <https://openreview.net/forum?id=AL1fq05o7H>.
- A. Gu, K. Goel, and C. Re. Efficiently modeling long sequences with structured state spaces. In *ICLR*, 2022. URL <https://openreview.net/forum?id=uYLFoz1v1AC>.
- E. Guha, R. Marten, S. Keh, N. Raoof, G. Smyrnis, H. Bansal, M. Nezhurina, J. Mercat, T. Vu, Z. Sprague, A. Suvarna, B. Feuer, L. Chen, Z. Khan, E. Frankel, S. Grover, C. Choi, N. Muennighoff, S. Su, W. Zhao, J. Yang, S. Pimpalgaonkar, K. Sharma, C. C.-J. Ji, Y. Deng, S. Pratt, V. Ramanujan, J. Saad-Falcon, J. Li, A. Dave, A. Albalak, K. Arora, B. Wulfe, C. Hegde, G. Durrett, S. Oh, M. Bansal, S. Gabriel, A. Grover, K.-W. Chang, V. Shankar, A. Gokaslan, M. A. Merrill, T. Hashimoto, Y. Choi, J. Jitsev, R. Heckel, M. Sathiamoorthy, A. G. Dimakis, and L. Schmidt. OpenThoughts: Data recipes for reasoning models, 2025. URL <https://arxiv.org/abs/2506.04178>.
- D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, et al. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence, 2024. URL <https://arxiv.org/abs/2401.14196>.

- Y. Hao, D. Angluin, and R. Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *TACL*, 10:800–810, 2022. doi: 10.1162/tacl_a_00490. URL <https://aclanthology.org/2022.tacl-1.46/>.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *ICLR*, 2021.
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models, 2022. URL <https://arxiv.org/abs/2203.15556>.
- C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekes, F. Jia, and B. Ginsburg. RULER: What’s the real context size of your long-context language models? In *COLM*, 2024. URL <https://openreview.net/forum?id=kIoBbc76Sy>.
- M. Y. Hu, J. Petty, C. Shi, W. Merrill, and T. Linzen. Between circuits and Chomsky: Pre-pretraining on formal languages imparts linguistic biases. In *ACL*, 2025. URL <https://aclanthology.org/2025.acl-long.478/>.
- S. Hu, Y. Tu, X. Han, G. Cui, C. He, W. Zhao, X. Long, Z. Zheng, Y. Fang, Y. Huang, X. Zhang, Z. L. Thai, C. Wang, Y. Yao, C. Zhao, J. Zhou, J. Cai, Z. Zhai, N. Ding, C. Jia, G. Zeng, D. Li, Z. Liu, and M. Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *COLM*, 2024. URL <https://openreview.net/forum?id=3X2L2Tfr0f>.
- M. Hutter. Learning curve theory, 2021. URL <https://arxiv.org/abs/2102.04074>.
- S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, S. L. Song, S. Rajbhandari, and Y. He. DeepSpeed Ulysses: System optimizations for enabling training of extreme long sequence transformer models, 2023. URL <https://arxiv.org/abs/2309.14509>.
- N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code, 2024. URL <https://arxiv.org/abs/2403.07974>.
- S. Jelassi, D. Brandfonbrener, S. M. Kakade, and E. Malach. Repeat after me: Transformers are better than state space models at copying. In *ICML*, 2024. URL <https://proceedings.mlr.press/v235/jelassi24a.html>.
- D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models, 2020. URL <https://arxiv.org/abs/2001.08361>.
- A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML*, 2020. URL <https://proceedings.mlr.press/v119/katharopoulos20a.html>.
- Kimi Team. Kimi Linear: An expressive, efficient attention architecture, 2025. URL <https://arxiv.org/abs/2510.26692>.
- T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, K. Toutanova, L. Jones, M. Kelcey, M.-W. Chang, A. M. Dai, J. Uszkoreit, Q. Le, and S. Petrov. Natural questions: A benchmark for question answering research. *TACL*, 7:452–466, 2019. URL <https://aclanthology.org/Q19-1026>.
- W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.
- Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-T. Yih, D. Fried, S. Wang, and T. Yu. DS-1000: A natural and reliable benchmark for data science code generation, 2022. URL <https://arxiv.org/abs/2211.11501>.
- N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahma, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, V. Graf, J. D. Hwang, J. Yang, R. Le Bras, O. Tafjord, C. Wilhelm, L. Soldaini, N. A. Smith, Y. Wang, P. Dasigi, and H. Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL <https://arxiv.org/abs/2411.15124>.

- A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models, 2022. URL <https://arxiv.org/abs/2206.14858>.
- Z. Li, H. Liu, D. Zhou, and T. Ma. Chain of thought empowers transformers to solve inherently serial problems. In *ICLR*, 2024. URL <https://openreview.net/forum?id=3EWTEy9MTM>.
- K. Ligett. Let’s stop leaving money on the table, 2026. URL <https://simons.berkeley.edu/events/lets-stop-leaving-money-table-richard-m-karp-distinguished-lecture>. Richard M. Karp Distinguished Lecture, Simons Institute.
- H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let’s verify step by step, 2023. URL <https://arxiv.org/abs/2305.20050>.
- B. Y. Lin, R. Le Bras, K. Richardson, A. Sabharwal, R. Poovendran, P. Clark, and Y. Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning, 2025. URL <https://arxiv.org/abs/2502.01100>.
- B. Liu, J. T. Ash, S. Goel, A. Krishnamurthy, and C. Zhang. Transformers learn shortcuts to automata. In *ICLR*, 2023a. URL <https://openreview.net/forum?id=De4FYqjFueZ>.
- J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *NeurIPS*, 2023b. URL <https://openreview.net/forum?id=1qvx610Cu7>.
- Y. Liu, K. Preechakul, K. Kuwarananchaoen, and Y. Bai. The serial scaling hypothesis. In *ICLR*, 2026. URL <https://openreview.net/forum?id=0bXB7KJn0B>.
- K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. Weld. S2ORC: The semantic scholar open research corpus. In *ACL*, 2020. URL <https://aclanthology.org/2020.acl-main.447>.
- C. London and V. Kanade. Pause tokens strictly increase the expressivity of constant-depth transformers. In *NeurIPS*, 2025. URL <https://openreview.net/forum?id=eG5oh811WZ>.
- A. Mallen, A. Asai, V. Zhong, R. Das, H. Hajishirzi, and D. Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories, 2022. URL <https://arxiv.org/abs/2212.10511>.
- A. Matton, T. Sherborne, D. Aumiller, E. Tommasone, M. Alizadeh, J. He, R. Ma, M. Voisin, E. Gilsonan-McMahon, and M. Gallé. On leakage of code generation evaluation datasets. In *Findings of EMNLP*, 2024. URL <https://aclanthology.org/2024.findings-emnlp.772/>.
- S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models, 2016. URL <https://arxiv.org/abs/1609.07843>.
- W. Merrill. Sequential neural networks as automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13, 2019. doi: 10.18653/v1/W19-3901. URL <https://aclanthology.org/W19-3901/>.
- W. Merrill. On the linguistic capacity of real-time counter automata, 2021. URL <https://arxiv.org/abs/2004.06866>.
- W. Merrill. *A Theory of the Computational Power and Limitations of Language Modeling Architectures*. PhD thesis, New York University, 2025. URL <https://lambdaviiking.com/assets/pdf/dissertation.pdf>.
- W. Merrill and A. Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *TACL*, 11:531–545, 2023. doi: 10.1162/tacl_a_00562. URL <https://aclanthology.org/2023.tacl-1.31/>.
- W. Merrill and A. Sabharwal. The expressive power of transformers with chain of thought. In *ICLR*, 2024. URL <https://openreview.net/forum?id=NjNG1Ph8Wh>.
- W. Merrill and A. Sabharwal. Exact expressive power of transformers with padding. In *NeurIPS*, 2025. URL <https://openreview.net/forum?id=01abxStFcy>.
- W. Merrill, J. Petty, and A. Sabharwal. The illusion of state in state-space models. In *ICML*, 2024. URL <https://openreview.net/forum?id=QZgo9JZpLq>.
- W. Merrill, S. Arora, D. Groeneveld, and H. Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. In *NeurIPS*, 2025. URL <https://openreview.net/forum?id=XUKUx7Xu89>.
- W. Merrill, H. Jiang, Y. Li, A. Lin, and A. Sabharwal. Why are linear RNNs more parallelizable?, 2026. URL <https://arxiv.org/abs/2603.03612>.

- E. J. Michaud. On neural scaling and the quanta hypothesis. <https://ericjmichaud.com/quanta/>, 2026. Blog post.
- E. J. Michaud, Z. Liu, U. Girit, and M. Tegmark. The quantization model of neural scaling. In *NeurIPS*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5b6346a05a537d4cdb2f50323452a9fe-Abstract-Conference.html.
- I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL <https://arxiv.org/abs/2410.05229>.
- T. M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, Department of Computer Science, 1980. URL https://www.cs.cmu.edu/~tom/pubs/NeedForBias_1980.pdf.
- M. Mohri. Rational transductors, 2026. URL <https://arxiv.org/abs/2602.07599>.
- MosaicML. LLM Foundry – jeopardy dataset, 2024. URL <https://github.com/mosaicml/llm-foundry>.
- Y. Nam, N. Fonseca, S. H. Lee, C. Mingard, and A. A. Louis. An exactly solvable model for emergence and scaling laws in the multitask sparse parity problem. In *NeurIPS*, 2024. URL <https://openreview.net/forum?id=cuWsR25bbI>.
- NVIDIA. Nemotron-H: A family of accurate and efficient hybrid mamba-transformer models, 2025. URL <https://arxiv.org/abs/2504.03624>.
- NVIDIA Team. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025a. URL <https://arxiv.org/abs/2508.14444>.
- NVIDIA Team. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for agentic reasoning, 2025b. URL <https://arxiv.org/abs/2512.20848>. Technical report.
- Olmo Team. 2 OLMo 2 furious, 2024. URL <https://arxiv.org/abs/2501.00656>.
- Olmo Team. Olmo 3, 2025. URL <https://arxiv.org/abs/2512.13961>.
- C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads, 2022. URL <https://arxiv.org/abs/2209.11895>.
- OpenThoughts-Agent Team. OpenThoughts-Agent, 2025. URL <https://www.open-thoughts.ai/blog/agent>.
- A. Pal, L. K. Umapathi, and M. Sankarasubbu. MedMCQA: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *CHIL*, 2022. URL <https://proceedings.mlr.press/v174/pal22a.html>.
- D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context, 2016. URL <https://arxiv.org/abs/1606.06031>.
- B. Peng, J. Quesnelle, H. Fan, and E. Shippole. YaRN: Efficient context window extension of large language models. In *ICLR*, 2024. URL <https://openreview.net/forum?id=wHBfxhZu1u>.
- B. Peng, R. Zhang, D. Goldstein, E. Alcaide, X. Du, H. Hou, J. Lin, J. Liu, J. Lu, W. Merrill, G. Song, K. Tan, S. Utpala, N. Wilce, J. S. Wind, T. Wu, D. Wuttke, and C. Zhou-Zheng. RWKV-7 “goose” with expressive dynamic state evolution. In *COLM*, 2025. URL <https://openreview.net/forum?id=ayB1PACN5j>.
- J. Pfau, W. Merrill, and S. R. Bowman. Let’s think dot by dot: Hidden computation in transformer language models. In *COLM*, 2024. URL <https://openreview.net/forum?id=NikbrdtYvG>.
- V. Pyatkin, S. Malik, V. Graf, H. Ivison, S. Huang, P. Dasigi, N. Lambert, and H. Hajishirzi. Generalizing verifiable instruction following, 2025. URL <https://arxiv.org/abs/2507.02833>.
- Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. <https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd&from=research.latest-advancements-list>, 2025. Blog post.
- Qwen Team. Qwen3.5: Towards native multimodal agents. <https://qwen.ai/blog?id=qwen3.5>, 2026. Blog post.
- A. Radford and K. Narasimhan. Improving language understanding by generative pre-training, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
- R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*, 36, 2024.

- C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. URL <https://arxiv.org/abs/1910.10683>.
- S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *ICML*, pages 18332–18346, 2022. URL <https://proceedings.mlr.press/v162/rajbhandari22a.html>.
- P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016. URL <https://aclanthology.org/D16-1264>.
- S. Reddy, D. Chen, and C. D. Manning. CoQA: A conversational question answering challenge. *TACL*, 7:249–266, 2019. URL <https://aclanthology.org/Q19-1016>.
- M. Reid, V. Zhong, S. Gururangan, and L. Zettlemoyer. M2D2: A massively multi-domain language modeling dataset. In *EMNLP*, 2022. URL <https://aclanthology.org/2022.emnlp-main.63>.
- D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *COLM*, 2024. URL <https://openreview.net/forum?id=Ti67584b98>.
- L. Ren, Y. Liu, Y. Lu, Y. Shen, C. Liang, and W. Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *ICLR*, 2025. URL <https://openreview.net/forum?id=bI1npVM4bc>.
- K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at scale. *AAAI*, 34(05):8732–8740, 2020.
- M. Sap, H. Rashkin, D. Chen, R. Le Bras, and Y. Choi. Social IQa: Commonsense reasoning about social interactions. In *EMNLP*, 2019. URL <https://aclanthology.org/D19-1454>.
- Y. Sarrof, Y. Veitsman, and M. Hahn. The expressive capacity of state space models: A formal language perspective. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=eV5YIrJPdy>.
- W. J. Savitch. Why it might pay to assume that languages are infinite. *Annals of Mathematics and Artificial Intelligence*, 8:17–25, 1993. URL <https://link.springer.com/article/10.1007/BF02451546>.
- D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models, 2019. URL <https://arxiv.org/abs/1904.01557>.
- I. Schlag, K. Irie, and J. Schmidhuber. Linear transformers are secretly fast weight programmers. In *ICML*. PMLR, 2021. URL <https://proceedings.mlr.press/v139/schlag21a.html>.
- R. Shao, A. Asai, S. Z. Shen, H. Ivison, V. Kishore, J. Zhuo, X. Zhao, M. Park, S. G. Finlayson, D. Sontag, T. Murray, S. Min, P. Dasigi, L. Soldaini, F. Brahma, W.-t. Yih, T. Wu, L. Zettlemoyer, Y. Kim, H. Hajishirzi, and P. W. Koh. DR Tulu: Reinforcement learning with evolving rubrics for deep research, 2025. URL <https://arxiv.org/abs/2511.19399>.
- N. Shazeer. GLU variants improve transformer, 2020. URL <https://arxiv.org/abs/2002.05202>.
- J. Siems, R. Grazi, K. Kalinin, H. Ballani, and B. Rahmani. Learning state-tracking from code using linear rnns, 2026. URL <https://arxiv.org/abs/2602.14814>.
- L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, V. Hofmann, A. H. Jha, S. Kumar, L. Lucy, X. Lyu, N. Lambert, I. Magnusson, J. Morrison, N. Muennighoff, A. Naik, C. Nam, M. E. Peters, A. Ravichander, K. Richardson, Z. Shen, E. Strubell, N. Subramani, O. Tafjord, P. Walsh, L. Zettlemoyer, N. A. Smith, H. Hajishirzi, I. Beltagy, D. Groeneveld, J. Dodge, and K. Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research, 2024. URL <https://arxiv.org/abs/2402.00159>.
- L. Strobl, W. Merrill, G. Weiss, D. Chiang, and D. Angluin. What formal languages can transformers express? a survey. *TACL*, 12:543–561, 2024. doi: 10.1162/tacl_a_00663. URL <https://aclanthology.org/2024.tacl-1.30/>.
- J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL <https://doi.org/10.1016/j.neucom.2023.127063>.
- Y. Sun, S. Hu, G. Zhou, K. Zheng, H. Hajishirzi, N. Dziri, and D. X. Song. OMEGA: Can LLMs reason outside the box in math? evaluating exploratory, compositional, and transformative generalization, 2025. URL <https://arxiv.org/abs/2506.18880>.
- R. S. Sutton. The bitter lesson. https://heartyhaven.github.io/files/bitter_lesson.pdf, 2019.

- M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them, 2022. URL <https://arxiv.org/abs/2210.09261>.
- A. Talmor, J. Herzig, N. Lourie, and J. Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *NAACL*, 2019. URL <https://aclanthology.org/N19-1421>.
- A. Terzic, N. Menet, M. Hersche, T. Hofmann, and A. Rahimi. Structured sparse transition matrices to enable state tracking in state-space models. In *NeurIPS*, 2025. URL <https://openreview.net/forum?id=RDbuSCWhad>.
- V. Vapnik. Principles of risk minimization for learning theory. In *NeurIPS*, pages 831–838, 1991. URL <https://proceedings.neurips.cc/paper/1991/hash/ff4d5fbbafdf976cfdc032e3bde78de5-Abstract.html>.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL <https://arxiv.org/abs/2406.01574>.
- J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. In *W-NUT*, 2017. URL <https://aclanthology.org/W17-4413/>.
- K. Wen, X. Dang, and K. Lyu. RNNs are not transformers (yet): The key bottleneck on in-context retrieval. In *ICLR*, 2025a. URL <https://openreview.net/forum?id=h3wbI8Uk1Z>.
- K. Wen, Z. Li, J. S. Wang, D. L. W. Hall, P. Liang, and T. Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape view. In *ICLR*, 2025b. URL <https://openreview.net/forum?id=m51BgoqvBP>.
- A. G. Wilson. Position: Deep learning is not so mysterious or different. In *ICML Position Paper Track*, 2025. URL <https://openreview.net/forum?id=42Au7FoD8F>.
- G. Yang, J. B. Simon, and J. Bernstein. A spectral condition for feature learning, 2024a. URL <https://arxiv.org/abs/2310.17813>.
- S. Yang and Y. Zhang. FLA: A triton-based library for hardware-efficient implementations of linear attention mechanism, 2024. URL <https://github.com/fla-org/flash-linear-attention>.
- S. Yang, B. Wang, Y. Shen, R. Panda, and Y. Kim. Gated linear attention transformers with hardware-efficient training. In *ICML*, 2024b. URL <https://proceedings.mlr.press/v235/yang24ab.html>.
- S. Yang, B. Wang, Y. Zhang, Y. Shen, and Y. Kim. Parallelizing linear transformers with the delta rule over sequence length. In *NeurIPS*, 2024c. URL <https://openreview.net/forum?id=y8Rm4VNRPH>.
- S. Yang, J. Kautz, and A. Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025a. URL <https://arxiv.org/abs/2412.06464>.
- S. Yang, Y. Shen, K. Wen, S. Tan, M. Mishra, L. Ren, R. Panda, and Y. Kim. PaTH attention: Position encoding via accumulating householder transformations. In *NeurIPS*, 2025b. URL <https://openreview.net/forum?id=ZB1HEeSvKd>.
- R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *ACL*, 2019. URL <https://aclanthology.org/P19-1472>.
- B. Zhang and R. Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.
- J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models, 2023. URL <https://arxiv.org/abs/2311.07911>.
- T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widayarsi, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions, 2024. URL <https://arxiv.org/abs/2406.15877>.
- J. Zuo, M. Velikanov, D. E. Rhaïem, I. Chahed, Y. Belkada, G. Kunsch, and H. Hacid. Falcon Mamba: The first competitive attention-free 7B language model, 2024. URL <https://arxiv.org/abs/2410.05355>.

J. Zuo, M. Velikanov, I. Chahed, Y. Belkada, D. E. Rhyem, G. Kunsch, H. Hacid, H. Yous, B. Farhat, I. Khadraoui, et al. Falcon-H1: A family of hybrid-head language models redefining efficiency and performance, 2025. URL <https://arxiv.org/abs/2507.22448>.

Table 9 Rough training throughput measurements run before launching the Olmo Hybrid pretraining run.

Model	# Heads	d_{model}	# Parameters	Training Throughput (TPS)
Olmo 3	32	4096	6.8B	8.0K
Olmo Hybrid	32	4096	7.7B	7.7K
	31	3968	7.4B	7.7K
	30	3840	7.0B	8.2K

A Training Olmo Hybrid

We elaborate on experiments and implementation details in the development of Olmo Hybrid.

A.1 Pretraining

Each Olmo Hybrid GDN head is sized proportionately to an Olmo 3 attention head. In line with standard mappings from attention to GDN¹⁰ this means that the size of the query and key becomes $d = 3/4 \cdot 128 = 96$. Similarly, the size of the value becomes $2d = 192$.

We used the Flash Linear Attention (Yang and Zhang, 2024) implementation of Gated DeltaNet. In particular, we used the default parameter implemented when instantiating single layer directly (as opposed to the model-level initialization).

Beyond the individual GDN heads, the overall architecture is roughly the same Olmo 3 7B (Olmo Team, 2025) with a few small tweaks to the architecture, learning rate schedule and training data:

- We removed two heads from the model to make Olmo 3 and Olmo Hybrid more comparable in parameter count and training throughput (see below).
- Rather than using the ad-hoc piecewise learning rate schedule from Olmo 3 7B (Olmo Team, 2025), we use a standard cosine decay to 10% of the maximum learning rate. However, the learning rate schedules still match closely for the majority of training, especially towards the beginning.
- For data, we use the improved data mix from Olmo 3 32B rather than the data mix from Olmo 3 7B—in preliminary experiments, we ran with the Olmo 3 7B data mix and saw similar trends in pretraining evaluation metrics toward the beginning of training.

The GDN architecture and 3:1 hybridization ratio were chosen based on early experiments at the scale of 1B parameters and 100B tokens. We replicate those early experiments as carefully controlled architecture ablations in Section 5.1.

The model was trained on 512 GPUs. At the beginning of training, these were H100s, but roughly halfway through pretraining we migrated to 512 B200s.

Training Throughput. We calibrated training throughput to be comparable to that of Olmo 3 in initial experiments by removing individual heads until it closely matched the transformer. Concretely, we benchmarked model size and throughput for several training configurations running on 128 H100s. As shown in Table 9, the hybrid model with 2 heads removed closely matches the Olmo 3 transformer in terms of parameters and slightly outperforms it in terms of training throughput. We therefore selected a hybrid architecture with 30 heads for the Olmo Hybrid 7B training run.

Training Stability. To estimate the stability of Olmo training runs, we compute a *spike score* as an objective measure. Concretely, we define the spike score as the percentage of values in a time series that are at least six standard deviations away from a rolling average of the last 128 values. We use spike score on the L2 norm of

¹⁰https://github.com/fla-org/flash-linear-attention/blob/f24317a6a4f513748cd7eb05818534ce66029957/fla/layers/gated_deltanet.py#L40

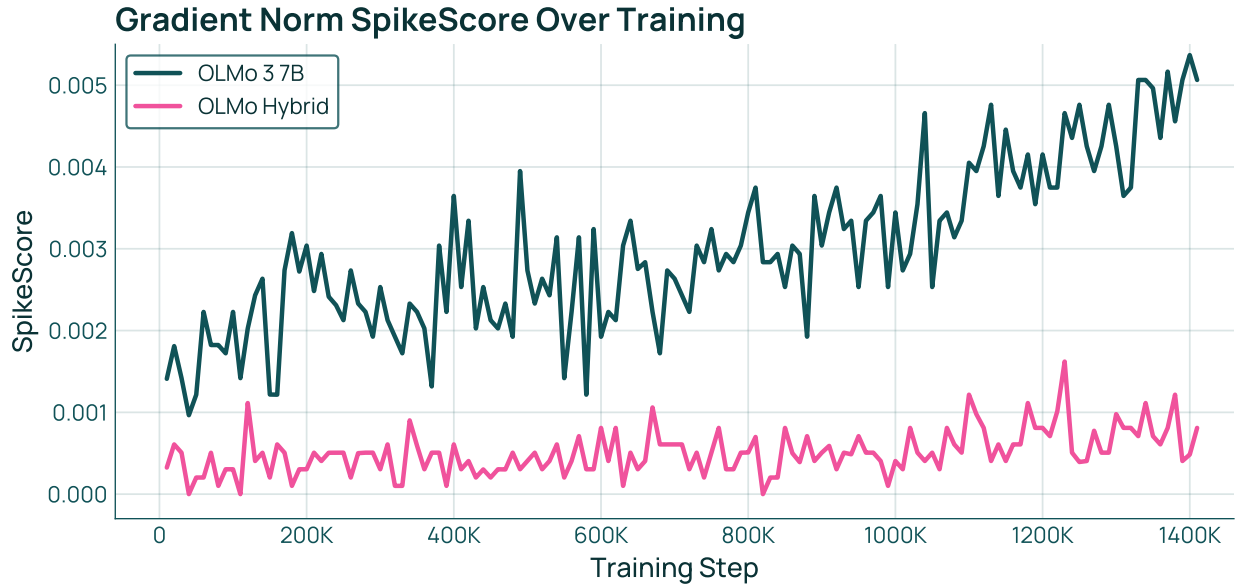


Figure 13 SpikeScore grows throughout training for Olmo 3, but it stays more stable when training Olmo Hybrid.

the gradient. This is a slight modification from the method used in Olmo Team (2024), leading to slightly higher scores.

Figure 13 shows that Olmo 3 exhibits a high and growing number of spikes in the gradient norm, while Olmo Hybrid shows a lower, flat trajectory. We take this as preliminary evidence that the Olmo Hybrid architecture may be more stable (i.e., more tolerant of large learning rates and noisy data) compared to Olmo 3.

A.2 Mid-Training and Long Context Extension

Mid-Training. We adapt our mid-training procedure from Olmo 3 (Olmo Team, 2025), using the Olmo 3 32B mid-training data (the Olmo 3 7B mid-training data mixture with additional light filtering applied). One notable change is a doubled batch size, motivated by recent insights into the relationship between learning rate and batch size (Merrill et al., 2025). Following the Olmo 3 32B recipe, we perform two independent mid-training runs on separate 100B token subsets of Dolma 3 Dolmino Mix and merge the resulting checkpoints.

Long Context Extension. After mid-training, we extend context length by continuing training on 100B tokens of Dolma 3 Longmino Mix. We compare two positional encoding strategies for this stage. The first is YaRN (Peng et al., 2024), which was also used for Olmo 3 (Olmo Team, 2025). YaRN modifies the RoPE frequency basis by partitioning dimensions according to frequency: low-frequency components (encoding longer-range position) are interpolated to cover the target context length, while high-frequency components (encoding local position) are left unchanged. An attention temperature factor corrects for the shift in attention logit magnitudes introduced by the interpolation. The second is DroPE (Gelberg et al., 2025), which removes RoPE entirely during long-context extension. The model then relies on the causal attention mask and any positional signal already captured in its weights. The motivation is that RoPE’s rotation frequencies, fit to shorter contexts during pretraining, can limit extrapolation to longer sequences; dropping them removes this constraint.

Both strategies produce strong long-context results for Olmo Hybrid (Table 3), but DroPE shows a clear advantage at the longest evaluation lengths (e.g., 85.0 vs. 76.9 on RULER 64k). We attribute this in part to the hybrid architecture: the GDN layers carry implicit positional information through their recurrent structure, so the attention layers are less dependent on explicit positional encodings like RoPE. We therefore adopt DroPE for the released Olmo Hybrid checkpoint.

To train at these longer sequence lengths, we implemented Ulysses-style context parallelism (Jacobs et al.,

2023) through both the attention and GDN layers. Ulysses distributes the sequence across devices and uses all-to-all communication to transpose from a sequence-parallel layout to a head-parallel layout before each layer. After the all-to-all, each device holds the full sequence for a subset of heads, which suffices for both attention (where each head attends independently) and GDN (where the recurrent state update in Definition 1 is also per-head). The short depthwise convolutions (kernel size 4) applied to the q , k , and v streams in GDN layers operate per-channel along the sequence dimension; since channels are partitioned across heads, the convolution weights must be sharded consistently with the head assignment on each device.

A.3 Post-Training

Table 10 Training hyperparameters for Olmo Hybrid 7B. Total tokens includes masked tokens (e.g. prompts). Think SFT total tokens is computed from the Olmo 3 baseline (45.4B) plus the $3 \times$ tool-use upscale: $45.4B \times (1 + 2 \times 2.47\%) = 47.6B$.

	7B Think SFT	7B Instruct SFT	7B Instruct DPO
Instances	2,932,239	2,153,716	259,922
Total Tokens	47.6B	3.4B	N/A
Batch Size	1M tokens	1M tokens	N/A
Learning Rate	2.5×10^{-5}	2.5×10^{-5}	1×10^{-6}
Num. GPUs	64	64	32
Max Sequence Length	32K	32K	16K
Epochs	2	2	1
Loss	—	—	DPO Norm ($\beta=5$)

Table 11 Differences between Olmo 3 and Olmo Hybrid SFT training configurations.

	Olmo 3	Olmo Hybrid
Data Parallel	HSDP (shard within node)	FSDP (full sharding)
Context Parallel	Ring (degree = 8)	Ulysses (degree = 2)
Activation Checkpointing	Selected modules (<code>feed_forward</code>)	Budget-based (0.1)

Table 10 summarizes the hyperparameters used for the Olmo Hybrid 7B model variants. Table 11 highlights the key configuration differences between the Olmo 3 and Olmo Hybrid SFT training setups, primarily involving changes to the parallelism strategy and activation checkpointing. The SFT models were trained with OLMo-core, and the DPO model was trained with Open-Instruct.

A.4 Evaluation Details

We evaluate Olmo Hybrid using the same evaluation suite as OLMo 3 (Olmo Team, 2025). Table 12 describes the base evaluation configuration and Table 13 describes the post-training evaluation configuration.

B Proofs: Expressive Power of Hybrid Models

Throughout this section, we assume by default that complexity classes (e.g., TC^0) refer to their FO-uniform variants (i.e., FO-uniform TC^0). We formalize *bounded precision* to mean logarithmic precision, i.e., on input sequences of length n , we compute our model with $c \log n$ bits of precision, for some c .

Theoretical analysis of transformers makes various assumptions about the types of attention allowed (Hao et al., 2022; Strobl et al., 2024). In our results, we will consider two types. First, we will consider unique-hard-attention transformers (UHATs), where attention can only attend to one unique position. Additionally, we will consider averaging-hard-attention transformers (AHATs), where attention weight is uniformly distributed over all positions that maximize the attention score. Our AHAT definition also allows masked pre-norm as in

Task		ICL	Format	Metric	Temp	Top-p	Max toks	P@k (n)	# sub	
Base Main Suite										
<i>Math</i>	GSM8K* (2021)	8 ^α	CoT EM	pass@k	0.6	0.6	512	1, 4 (8)	-	
	GSM Symbolic* (2024)	8 ^α	CoT EM	pass@k	0.6	0.6	512	1, 4 (8)	3	
	MATH 500* (2022; 2023)	4 ^α	CoT EM	pass@k	0.6	0.6	1024	1, 16 (32)	-	
<i>Code</i>	HumanEval* (2021)	3	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-	
	MBPP* (2021)	3	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-	
	BigCodeBench* (2024)	3	Code Exec	pass@k	0.6	0.6	1280	1 (5)	-	
	DS 1000* (2022)	3	Code Exec	pass@k	0.6	0.6	1024	1 (5)	-	
	Deepseek LeetCode* (2024)	0	Code Exec	pass@k	0.6	0.6	512	1, 16 (32)	-	
	MultiPL-E HumanEval* (2022)	0	Code Exec	pass@k	0.6	0.6	1024	1, 16 (32)	6	
	MultiPL-E MBPP* (2022)	0	Code Exec	pass@k	0.6	0.6	1024	1, 16 (32)	6	
	ARC (2018)	5	MC	Acc	-	-	-	-	2	
	MMLU STEM (2021)	5	MC	Acc	-	-	-	-	19	
<i>STEM QA</i>	MedMCQA* (2022)	5	MC	Acc	-	-	-	-	-	
	MedQA* (2021)	5	MC	Acc	-	-	-	-	-	
	SciQ* (2017)	5	MC	Acc	-	-	-	-	-	
	MMLU Humanities (2021)	5	MC	Acc	-	-	-	-	13	
	MMLU Social Sci. (2021)	5	MC	Acc	-	-	-	-	12	
<i>Non-STEM QA</i>	MMLU Other (2021)	5	MC	Acc	-	-	-	-	14	
	CSQA (2019)	5	MC	Acc	-	-	-	-	-	
	PiQA (2020)	5	MC	Acc	-	-	-	-	-	
	SocialIQA (2019)	5	MC	Acc	-	-	-	-	-	
	DROP Gen2MC* (introduced in Olmo Team (2025); 2019)	5	MC	Acc	-	-	-	-	-	
	Jeopardy Gen2MC* (introduced in Olmo Team (2025); 2024)	5	MC	Acc	-	-	-	-	-	
	NaturalQs Gen2MC* (introduced in Olmo Team (2025); 2019)	5	MC	Acc	-	-	-	-	-	
	SQuAD Gen2MC* (introduced in Olmo Team (2025); 2016)	5	MC	Acc	-	-	-	-	-	
	CoQA Gen2MC* (introduced in Olmo Team (2025); 2019)	0 [†]	MC	Acc	-	-	-	-	-	
	Basic Skills* (introduced in Olmo Team (2025))	5	MC	Acc	-	-	-	-	6	
	<i>GenQA</i>	HellaSwag (2019)	5	RC _{per-char}	Acc	-	-	-	-	-
		WinoGrande (2020)	5	RC _{none}	Acc	-	-	-	-	-
		Lambada (2016)	0	RC _{per-char}	Acc	-	-	-	-	-
Basic Skills* (introduced in Olmo Team (2025))		5	RC _{per-token}	Acc	-	-	-	-	6	
DROP (2019)		5	GenQA	F1	0	1	100	-	-	
Jeopardy (2024)		5	GenQA	F1	0	1	50	-	-	
NaturalQs (2019)		5	GenQA	F1	0	1	50	-	-	
SQuAD (2016)		5	GenQA	F1	0	1	50	-	-	
CoQA (2019)		0 [†]	GenQA	F1	0	1	50	-	-	
Base Held-out Suite										
MMLU Pro (2024)	5	MC	Acc	-	-	-	-	-	13	
LBPP* (2024)	0	Code Exec	pass@k	0.6	0.6	4096	1 (32)	-		
Deepmind Math* (2019)	5	CoT EM	pass@k	0.6	0.6	2048	1 (1)	-		
BigBench Hard (2022)	3	CoT EM	Acc	0.6	0.6	512	1 (1)	55		

Table 12 Details of the base evaluation suite, as used in OLMo 3 (Olmo Team, 2025). Tasks were formatted as multiple-choice (MC), rank choice (RC), short-form generative (GenQA), chain-of-thought with exact-match scoring (CoT EM), or code execution (Code Exec). We use * to indicate additions to the OLMo 2 (Olmo Team, 2024) base suite, † for tasks with few-shot examples already specified within each instance, and ^α for tasks with human-written few-shot examples.

Merrill and Sabharwal (2025, Section 2.1). Our constructions for UHATs also work for AHATs, but we state them for UHATs for more generality.

We first establish the negative side of the main theorems (Theorem 1 and corollary 3.1) via core lemmas in Section B.1. We then complete the proofs by giving explicit constructions for hybrid models solving these problems. Finally, we turn to giving a complete padded characterization.

B.1 Limitations of Transformers and RNNs

We will now formalize the limitations of transformers and RNNs on several problems of interest via two core lemmas. The same problems are inexpressible for these architectures for different reasons. In the case of transformers, it is because these problems are inherently sequential, which we formalize via circuit complexity. In the case of RNNs, it is because they require remembering a significant amount of information from the prefix of the string, which we formalize using communication complexity.

We first establish the inability of fixed-depth transformers (both AHAT and UHAT) to express the key

Task	Format	Metric	Temp	Top-p	Ans. Extract	Max Toks	N	# Sub
Chat Suite								
IF Eval (2023)	CoT	Custom	0.6	0.95	Custom	32768	1	-
IFBench (2025)	CoT	Custom	0.6	0.95	Custom	32768	1	-
MATH 500 (2022; 2023)	CoT EM	EM Flex	0.6	0.95	Minerva	32768	1	-
AIME 2024*	CoT EM	EM Flex	0.6	0.95	Minerva	32768	32	-
AIME 2025*	CoT EM	EM Flex	0.6	0.95	Minerva	32768	32	-
Omega Math (2025)	CoT EM	EM Flex	0.6	0.95	Custom Regexes	32768	1	55
HumanEval+ (2023b)	CoT Code	pass@1	0.6	0.95	Split on ““	32768	10	-
MBPP+* (2023b)	CoT Code	pass@1	0.6	0.95	Split on ““	32768	10	-
LiveCodeBench v3* (2024)	CoT Code	pass@1	0.6	0.95	Split on ““	32768	10	-
ZebraLogic* (2025)	CoT JSON	Custom	0.6	0.95	Custom JSON	32768	1	-
BigBench-Hard (2022)	CoT EM	EM Flex	0.6	0.95	Custom Regex	32768	1	23
GPQA* (2024)	CoT MC	Acc	0.6	0.95	Custom Regex	32768	1	-
MMLU (2021)	CoT MC	Acc	0.6	0.95	Custom Regex	32768	1	57
PopQA (2022)	CoT MC	Acc	0.6	0.95	EM Recall	32768	1	-
Alpaca Eval v2 (2024)	CoT	Winrate	0.6	0.95	-	32768	1	-

Table 13 Details of the post-training evaluation suite, as used in OLMo 3 (Olmo Team, 2025). We mark tasks with * to indicate new additions compared to the OLMo 2 suite (Olmo Team, 2024). All evaluation generations have thinking traces (text between `<think>...</think>`) stripped before passing to the answer scorer. We use zero-shot setting for all metrics.

problems mentioned in Theorem 1 and Corollary 3.1:

Lemma 1 *Assuming $TC^0 \neq NC^1$, fixed-depth transformers cannot solve problems that are NC^1 -hard under FO reductions, which includes the A_5 word problem, state-based recall over 5 variables, and formula evaluation. This holds even with polynomial padding.*

Proof. This follows from the fact that fixed-depth AHATs, UHATs, and softmax transformers (even with padding) are in TC^0 (Merrill and Sabharwal, 2023; Chiang, 2025). If transformers could solve a problem hard for NC^1 under FO reductions, then any NC^1 problem could be solved in TC^0 by composing an AC^0 circuit for the FO reduction with the TC^0 circuit for the complete problem. This would imply $TC^0 = NC^1$, which contradicts the given premise.

We now justify that each of the mentioned problems is NC^1 -hard:

- The NC^1 completeness of the A_5 word problem is a fundamental result (Barrington, 1986). This holds even for the variant where the input string is a sequence of transpositions over 5 elements and the output is the number that 1 gets mapped to (Merrill et al., 2024, Section 3).
- To show state-based recall is NC^1 -complete, we construct an FO reduction from the transposition variant of the A_5 word problem as follows. Given a sequence of transpositions w , instantiate a fixed-length list where the first entry is 1 and others are 0. Then simply copy over the list of transpositions from the A_5 instance. By construction, the output reconstructs the number that w maps 1 to.
- The NC^1 -hardness of formula evaluation over booleans or integers is straightforward. Moreover, evaluating boolean formulas is also NC^1 -complete (Buss, 1987), whereas evaluating integer formulas is PNC^1 -complete (Caussinus et al., 1998). \square

Next we formalize the memory limitations of RNNs (linear or nonlinear) using standard techniques from communication complexity:

Lemma 2 *Log-precision RNNs (including DeltaNet) cannot solve problems with $\Omega(n)$ communication complexity, which includes recall, state-based recall, and formula evaluation in Polish notation. This result holds even with polynomial padding.*

Proof. It is natural that the bounded state size of RNNs restricts their expressive power (Jelassi et al., 2024).

Log-precision RNNs have a hidden state of size $O(\log n)$, and thus any problem that requires passing $\Omega(n)$ bits from prefix to suffix will not be expressible.

We next show that each of the listed problems have $\Omega(n)$ communication complexity, i.e., require passing this many bits from prefix to suffix when processing via streaming:

- It is natural that recall problems (e.g., recognizing ww) have $\Omega(n)$ communication complexity.
- State-based recall can be reduced to recall by a string homomorphism that simply deletes the state tracking tokens. Thus, it also requires $\Omega(n)$ communication complexity.
- Finally, evaluating formulas in Polish notation has a communication complexity of $\Omega(n)$ (Merrill, 2021, Theorem 6).

With polynomial padding tokens, precision remains logarithmic in the input sequence length:

$$O(\log(n^c)) = O(c \log n) = O(\log n).$$

Thus, we are still restricted to remembering at most $O(\log n)$ bits of an input prefix. \square

Thus, assuming both $\text{TC}^0 \neq \text{NC}^1$ and log precision, both transformers and linear RNNs cannot solve state-based recall, the permuted A_5 word problem, or formula evaluation. As a side note, the same communication complexity argument also shows that RNNs cannot even recognize recall languages like ww and ww^R , which have communication complexity $\Omega(n)$. These simple languages are, however, in uniform AC^0 and can also be easily recognized by fixed-depth AHATs (Strobl et al., 2024).

B.2 Power of Hybrid Models

We now turn to proving the main results that hybrid models can solve problems beyond the capabilities of both transformers and RNNs. In this section, we consider a hybrid model where the transformer component is an AHAT, though the construction also goes through with UHAT blocks as long as there is at least one linear RNN layer prior to it. The inexpressibility part of the result applies to both UHAT and AHAT. Assume the initial pointer values p_1, \dots, p_5 are encoded in unary (binary encoding is discussed after the following result and its proof) for the state-based recall problem.

Theorem 1: State-Based Recall Separation

There exists a hybrid model (GDN with negative eigenvalues + averaging-hard attention) that solves state-based recall, with just one alternation between layer types, in either order. In contrast, no transformer or RNN can express this problem, assuming $\text{TC}^0 \neq \text{NC}^1$ for transformers.

Proof. Lemma 1 shows transformers cannot solve state-based recall assuming $\text{TC}^0 \neq \text{NC}^1$. Similarly, Lemma 2 shows log-precision RNNs cannot express state-based recall unconditionally. We now describe how hybrid models with a single alternation can express pointer-based recall, considering each order separately.

(GDN + Attention) First, we use GDN to read the pointers p_1, \dots, p_5 into a residual stream cell, which can be done for either unary- or binary-encoded pointers. As mentioned in the main text, the hybrid model can capture state-based recall by first composing permutations over the pointers with a GDN block and then using attention to retrieve the value at p_1 . Since each transposition can be expressed as an identity + rank 1 matrix, it can directly be implemented by GDN (Grazzi et al., 2025). In the attention layer that implements recall, we first use layer-norm to project the final value of p_1 onto the unit sphere, which we denote $\phi(p_1)$. At each input bit x_i , we also compute the projection $\phi(i)$ using standard tricks with NoPE (Merrill and Sabharwal, 2024, 2025). We then implement the recall by attending with query $\phi(p_1)$ over keys $\phi(i)$ and values x_i . Since attention is maximized exactly when $p_1 = i$, we will retrieve the correct bit x_{p_1} . Thus, we have constructed a hybrid model of DeltaNet layers followed by attention layers that solves state-based recall.

(*Attention + GDN*) The order of layers is reversed, but the overall idea remains similar. We first use an AHAT block to compute, for each pointer k , quantities $\phi(p_k/i)$ and $\phi(1/i)$, from which $\phi(p_k)$ can be computed. This is possible with averaging-hard attention (hence reliance on AHAT) because p_k is encoded in unary. Next, we use another layer of 5 attention heads to retrieve each value x_{p_k} from each pointer p_k using query $\phi(p_k)$, key $\phi(i)$, and value x_i . Finally, we use GDN to implement transposition composition over the values x_{p_1}, \dots, x_{p_5} , rather than over the pointers. Thus, in either order, one alternation allows a hybrid model to express state-based recall. \square

If GDN precedes attention, then this construction works for binary-encoded pointers in addition to unary-encoded pointers, as well as for UHAT blocks. It follows that, with more than one alternation, hybrid models (regardless of UHAT or AHAT) can also handle binary-encoded pointers, since this setting subsumes the one where GDN precedes attention.

B.3 Padded Hybrid Models

First, we describe padding in a little more detail. Given a function $t : \mathbb{N} \rightarrow \mathbb{N}$, a model is run with $t(n)$ padding as follows. For any input w , we append $t(n)$ padding tokens (\square) to get a padded input $w\square^{t(|w|)}$. We then interpret the prediction at the final token of this padded input as the prediction for w . We say that a language L can be recognized by a transformer with polynomial padding if there exists c and a transformer T such that T recognizes L with n^c padding.

We show that polynomially-padded hybrid models (with averaging-hard attention) can capture all of NC^1 :

Theorem 3: Padded Hybrid Models

With polynomial padding tokens, fixed-depth hybrid models (averaging-hard attention + GDN with negative eigenvalues) can recognize any language in FO-uniform NC^1 .

Proof. Let $L \in \text{NC}^1$. Recognizing L can be decomposed to implementing an FO reduction to an NC^1 -complete problem L' . Let L' be the transposition variant of the S_5 word problem described by Merrill et al. (2024, Section 3.1). We use the fact that every language $L \in \text{NC}^1$ is FO-reducible to L' . That is, any $w \in \Sigma^n$ can be mapped via an FO reduction to a new sequence u of length n^k such that $\prod_{i=1}^{n^k} u_i = 1$ if and only if $w \in L$. We use a block of transformer layers to implement the FO reduction from w to u . At this point, we have n^k padding tokens where token i encodes u_i . Next, we can construct a fixed-depth GDN with negative eigenvalues that recognizes L' (Grazzi et al., 2025), which is possible because each element in the transition monoid can be written as an identity + rank 1 operator. Thus, by composition, following Lemma 3 of Merrill and Sabharwal (2025), this hybrid model will accept iff $u \in L'$. By construction, $u \in L'$ iff $w \in L$. Thus, given an arbitrary language $L \in \text{NC}^1$, we have constructed a hybrid model that recognizes L with n^c padding, where c is fixed for each L . \square

Based on recent work, Theorem 3 can be naturally extended to give the stronger lower bound of FO-uniform PNC^1 using the fact that GDN with negative eigenvalues can represent PNC^1 -complete problems (Caussinus et al., 1998; Merrill et al., 2026). This implies padded hybrid models can solve some additional problems not known to be in NC^1 , such as simulated weighted automata or evaluating formulas over integers. Moreover, nonuniform PNC^1 holds as an upper bound for hybrid models (Merrill et al., 2026). Thus, modulo details about uniformity, PNC^1 represents an exact expressivity characterization for padded hybrid models.

Moreover, recent related work has studied hybrid models that mix transformers with weighted transducers, approaching such models from the perspective of expressive power and learning theory (Mohri, 2026). Combined with the link between linear RNNs and weighted automata (Merrill et al., 2026), there is a close connection between these models and hybrid models that mix transformers and linear RNNs, with both architectures falling in PNC^1 and capable of expressing PNC^1 -complete problems.

C Additional Details: Synthetic Evaluations

This appendix documents the experimental setup used for the synthetic evaluation curves in Section 3.5, including model architectures, the hyperparameter search protocol, curricula, and the exact numbers plotted.

C.1 Tasks and Evaluation Protocol

All tasks are generated online (no fixed dataset) by sampling code-like strings and training models as next-token predictors.

Recall. A bit array of size m is instantiated, followed by a query of the form `assert bits[i] == _`. The model must output the correct bit. We evaluate across $m \in \{4, 8, 16, 32, 64, 128\}$.

State Tracking. Variables are initialized and updated by n swaps/assignments, followed by a query `assert v == _`. We evaluate across $n \in \{4, 8, 16, 32, 64, 128\}$.

State-Based Recall. A bit array of size m is instantiated; variables are initialized to indices in $[0, m-1]$ and then updated by n swaps. Finally, the model must answer `assert bits[v] == _` for a queried variable v . In the default state-based recall setting used in Section 3.5, we set $m = n$ and evaluate across $n \in \{4, 8, 16, 32, 64, 128\}$.

Metric. We report next-token accuracy on the final answer token (the token immediately following the last `==`). Each reported accuracy is computed over 256 freshly generated evaluation samples at the specified difficulty.

C.2 Model Architectures

To address potential confounds from over-parameterization on shorter sequences, we use a compact, standardized architecture across all three models. They share the same base width and depth, differing only in their sequence-mixing layers (full attention vs. GDN vs. hybrid). Table 14 summarizes the shared hyperparameters and the model-specific settings.

Table 14 Architecture hyperparameters used for synthetic evaluations.

Shared hyperparameters	Value
Layers (L)	4
Model width (d_{model})	256
FFN intermediate size (d_{ff})	1024
Attention heads (H)	4
Head dimension (d_{head})	64
Max positions	1024 (Recall) / 4096 (State Tracking, State-based Recall)
Transformer	full softmax attention in all layers
Linear RNN (neg. eig.)	GatedDeltaNet with <code>allow_neg_eigval=True</code>
Linear RNN (pos. eig.)	GatedDeltaNet with <code>allow_neg_eigval=False</code>
Hybrid (neg. eig.)	first 3 layers are GDN with <code>allow_neg_eigval=True</code> ; last layer is full attention
Hybrid (pos. eig.)	first 3 layers are GDN with <code>allow_neg_eigval=False</code> ; last layer is full attention

C.3 Training Details

We train with bf16 using the HuggingFace **Trainer** (AdamW). To ensure robust comparisons, we standardize fundamental training parameters across all tasks: a batch size of 32, gradient accumulation of 1, and a warmup period of 250 steps. Each experiment is conducted on a single H100 GPU.

Because model performance on these synthetic algorithmic tasks is highly sensitive to the learning rate, scheduler, and weight initialization, we conduct a systematic hyperparameter sweep rather than relying on arbitrary fixed values.

Recall and State Tracking Sweep. For the Recall and State Tracking tasks, we randomly sample 10 configurations from a grid of common learning rates (10^{-4} , $3 \cdot 10^{-4}$, 10^{-3}) and schedulers (cosine, constant). Models are trained for 50,000 steps (Recall) and 20,000 steps (State Tracking).

State-Based Recall Sweep. State-based recall is a harder compositional task and exhibits high sensitivity to initialization. Initial trials indicated that a learning rate of 10^{-3} was unstable, causing all architectures to fail to learn. We therefore refined our search space to learning rates of $\{10^{-4}, 3 \cdot 10^{-4}\}$ and schedulers $\in \{\text{cosine, constant}\}$. For each of the 4 combinations, we run 5 different random seeds across all three model types, totaling 120 runs. Models on this task are trained for up to 200,000 steps.

Curricula. To facilitate learning on these complex tasks, we employ task-specific curriculum strategies. For **State Tracking**, we use a deterministic, time-based step schedule, advancing the difficulty n through $\{8, 16, 32, 64\}$ at fixed cumulative step milestones (500, 1,500, 3,500, and 7,500 steps, respectively). For **Recall**, the task does not require a curriculum and is trained directly on a fixed bit-array size of $m = 128$. For **State-Based Recall**, we use a hybrid threshold-and-budget curriculum. The model starts at difficulty $n = 8$ and advances through $n \in \{8, 16, 32, 64\}$ if it achieves a 0.95 accuracy threshold (evaluated every 100 steps over a hold-out batch of 256 samples). To prevent stalling, the curriculum also forces an advancement if a step budget is exhausted (10,000 steps for $n = 8$; 30,000 steps for subsequent levels).

Data Generation. All tasks are framed as next-token prediction over Python-like execution traces. For **State Tracking**, we follow the method in Siems et al. (2026), where the generator inserts intermediate `assert` statements to explicitly reveal parts of the evolving program state across 5 variables. These reveals occur at fixed, difficulty-dependent intervals. For **Recall**, the sequence simply consists of a bit-array definition followed by a direct query, without any intermediate state updates. For **State-Based Recall**, which composes state tracking and retrieval, we use intermediate `assert` statements but further strengthen the supervision and discourage pattern-matching through two techniques: first, we randomize the spacing between `assert` statements across examples (sampling uniformly from powers of 2 up to n); second, we mix in a fraction of strict examples (20%) that omit intermediate `assert` statements entirely, forcing the model to answer the final query.

No-Negative-Eigenvalue Ablation. In addition to the main three-model comparison, we evaluate ablated linear-RNN and hybrid variants in which the GDN blocks disallow negative eigenvalues by setting `allow_neg_eigval=False`. These ablations use the same task definitions, metrics, and curricula as the main experiments.

Run Selection Criteria. To report the final numbers in Section 3.5, we select the best run for each model and task using a strict, standardized criterion: we identify the run that achieves the maximum accuracy on the hardest difficulty setting (maximum sequence length / max steps) at the final evaluation step. In the event of a tie, we break the tie by looking at the accuracy of the next longest length, continuing until a clear best run is isolated.

To ensure full reproducibility and transparency, the complete set of runs for our hyperparameter sweeps can be viewed on Weights & Biases at: <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval>.

WandB Naming Conventions. Note that the task names in the WandB run logs differ slightly from the terminology used in this paper. When navigating the logs, please use the following mapping:

- **Recall** is labeled as `retrieval` (e.g., `retrieval_transformer_lr1e-4_bs32_cosine`). The relevant evaluation metric is the `eval_normal` field.
- **State Tracking** is labeled as `state_tracking`. The relevant evaluation metric is the `eval_strict` field.

- **State-Based Recall** is labeled as `ptr` (e.g., `ptr-transformer-lr3e-4-cosine-seed8`). The relevant evaluation metric is the `eval_strict` field.

The no-negative-eigenvalue ablation projects use the same task naming, but are stored in separate projects whose names end with `no-neg-eigval`.

The exact numbers plotted in Figure 7 and detailed in Tables 15–17 are drawn from the following selected best runs:

State Tracking.

- **Transformer:** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/925yyp3l>
- **Linear RNN (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/i2t6vfgc>
- **Hybrid (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/7oxbr81u>
- **Linear RNN (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/8hc3oeoq>
- **Hybrid (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/2hsy3ksf>

Recall.

- **Transformer:** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/dattbjr0>
- **Linear RNN (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/570kpuuh>
- **Hybrid (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/5eljl1aeq>
- **Linear RNN (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/phcklxl1r>
- **Hybrid (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/elbinpt6>

State-Based Recall.

- **Transformer:** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/9rexm8xm>
- **Linear RNN (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/jnlavjjn>
- **Hybrid (neg. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/oqy3t8n3>
- **Linear RNN (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/2onzuhs7>
- **Hybrid (pos. eig.):** <https://wandb.ai/ai2-llm/0lmo-Hybrid-synth-eval/runs/zn13y4j1>

Table 15 State tracking accuracy (varying number of updates n).

n	Transformer	Linear RNN (neg. eig.)	Linear RNN (pos. eig.)	Hybrid (neg. eig.)	Hybrid (pos. eig.)
4	0.51172	1.00000	1.00000	1.00000	1.00000
8	0.27734	1.00000	1.00000	1.00000	1.00000
16	0.17188	1.00000	0.97266	1.00000	1.00000
32	0.23047	1.00000	0.64453	1.00000	1.00000
64	0.19922	1.00000	0.21484	1.00000	0.96094
128	0.23047	1.00000	0.22266	1.00000	0.36328

Table 16 Recall accuracy (varying bit-array size m).

m	Transformer	Linear RNN (neg. eig.)	Linear RNN (pos. eig.)	Hybrid (neg. eig.)	Hybrid (pos. eig.)
4	1.00000	1.00000	1.00000	1.00000	1.00000
8	1.00000	1.00000	1.00000	1.00000	1.00000
16	1.00000	1.00000	1.00000	1.00000	1.00000
32	1.00000	1.00000	1.00000	1.00000	1.00000
64	1.00000	0.83203	0.80078	1.00000	1.00000
128	0.96484	0.67578	0.67188	1.00000	1.00000

Table 17 State-based recall accuracy (varying number of swaps n , with $m = n$).

n	Transformer	Linear RNN (neg. eig.)	Linear RNN (pos. eig.)	Hybrid (neg. eig.)	Hybrid (pos. eig.)
4	0.82031	1.00000	0.76953	1.00000	0.85938
8	0.75000	1.00000	0.99219	1.00000	0.99219
16	0.73828	1.00000	0.85547	1.00000	0.93750
32	0.73828	1.00000	0.64453	1.00000	0.68750
64	0.62891	0.78125	0.59766	1.00000	0.61328
128	0.54297	0.63672	0.57422	1.00000	0.57031

D Scaling and Ablation Experiments: Details and Additional Results

D.1 Additional Results

D.1.1 Additional Scaling Law Results

This section supplements Section 4.1 with the full scaling law coefficient table with confidence intervals (Table 18), compute-optimal loss predictions across all three architectures and compute budgets (Table 19), projected token savings by scale (Table 20), and scaling law residual diagnostics (Figure 14).

Scaling Coefficients. Table 18 reports the full Chinchilla scaling law coefficients for all three architectures under both the unconstrained fit (all five parameters free) and the fixed-exponent fit ($\alpha = \beta = 0.22$ shared; only E , A , B refit), together with 95% bootstrap confidence intervals. The unconstrained fit yields wide CIs due to high joint uncertainty when exponents are free; the fixed-exponent fit tightens the CIs considerably. The clearest signal is in B (data efficiency): the hybrid’s $B = 83.7$ (CI [80.2, 87.1]) is robustly lower than the transformer’s 94.9 (CI [88.7, 102.0]), with non-overlapping CIs. The parameter coefficient A slightly favors the hybrid (70.1 vs. 71.8) but the CIs overlap. In the fixed-exponent fit the ordering of the irreducible loss E reverses relative to the unconstrained fit: the transformer achieves a lower E (1.569, CI [1.54, 1.60]) than the hybrid (1.597, CI [1.58, 1.61]), suggesting the apparent advantage in E for the hybrid in the unconstrained fit was a fitting artifact. The scaling exponents α and β are statistically indistinguishable across architectures—the CIs broadly overlap in the unconstrained fit—consistent with the theoretical prediction that expressivity shifts efficiency constants without altering exponents (Section 4.2).

Table 19 reports compute-optimal model sizes, dataset sizes, and predicted losses for all three architectures across compute budgets from 10^{18} to 10^{23} FLOPs. The pure GDN achieves modest but consistent loss improvements ($\Delta \approx -0.02$ to -0.05) across scales, though these are smaller than those of the hybrid model, reflecting its weaker data efficiency B despite lower parameter coefficient A . The compute-optimal allocation for each architecture is derived from the fitted scaling laws as described in Section D.2.

Table 20 reports the projected token requirements and savings factors at a discrete set of model scales, corresponding to Figure 10(b) in the main text.

Architecture	E	A	α	B	β	a_{opt}	b_{opt}	R^2
<i>Free fit (all five parameters fit independently)</i>								
Olmo 3	1.65 [1.11, 1.87]	108.83 [21.2, 448.7]	0.25 [0.14, 0.34]	83.45 [29.6, 495.0]	0.21 [0.15, 0.30]	0.46 [0.33, 0.68]	0.54 [0.32, 0.67]	0.9976
Hybrid	1.60 [1.25, 1.72]	71.14 [24.8, 145.3]	0.23 [0.15, 0.27]	81.72 [37.4, 219.6]	0.22 [0.18, 0.27]	0.49 [0.41, 0.64]	0.51 [0.36, 0.59]	0.9993
Pure GDN	1.46 [1.16, 1.75]	36.24 [19.0, 113.3]	0.18 [0.14, 0.26]	102.72 [52.1, 214.8]	0.23 [0.19, 0.26]	0.55 [0.45, 0.65]	0.45 [0.35, 0.55]	0.9994
<i>Fixed-exponent fit ($\alpha = \beta = 0.22$ shared; only E, A, B refit)</i>								
Olmo 3	1.55 [1.52, 1.58]	66.63 [61.9, 69.4]	0.22	94.85 [89.3, 101.4]	0.22	0.50	0.50	0.9975
Hybrid	1.58 [1.56, 1.59]	65.09 [62.8, 67.3]	0.22	83.65 [79.9, 87.0]	0.22	0.50	0.50	0.9993
Pure GDN	1.61 [1.59, 1.62]	62.38 [60.5, 64.8]	0.22	90.80 [87.4, 93.3]	0.22	0.50	0.50	0.9994

Table 18 Fit Chinchilla scaling law parameters for $L(N, D) = E + A/N^\alpha + B/D^\beta$. 95% bootstrap CI shown below each estimate. **Top:** unconstrained fit where all five parameters are fit independently per architecture. **Bottom:** fixed-exponent fit where $\alpha = \beta = 0.22$ (in gray cells) (approximately the mean of the unconstrained estimates) are shared across architectures; only E, A, B are refit. Equal exponents imply $a_{\text{opt}} = b_{\text{opt}} = 0.5$.

Fit Quality. The R^2 values from Table 18 (0.998 for Olmo 3, 0.999 for the hybrid model, and 0.999 for the linear RNN) indicate that the power-law form fits the observed data well. Figure 14 provides additional diagnostics for the quality of the fits. Panel (a) shows (signed) relative prediction errors $((L - \hat{L})/L) \times 100\%$ as a function of predicted loss \hat{L} , for all data points across all model sizes. Residuals are small (within $\pm 1\%$) and scattered around zero, confirming that the Chinchilla parametric form is an adequate model for all three architectures. Panel (b) reports the mean absolute relative error grouped by model size, showing that fit quality is consistent across the full range of scales (60M–1B parameters).

D.1.2 Ablation Results

This section supplements Section 5.1 with additional scaling plots across all tested architectures (Figure 16) and per-domain BPB evaluations for all configurations at all scales (Table 21).

Scaling Trends of All Tested Architectures. Figure 16 plots the scaling of the OLMOBASEVAL average score against training FLOPs for all architectures. Notably, the 7:1 configuration also shows very favorable scaling behavior, likely owing to the lower FLOP counts from having fewer transformer layers. The per-domain breakdown (Figure 16(c)–(e)) shows that the advantage of hybrid models is present across Math, Code, and QA. Fine-grained scaling plots broken down by ablation group are provided in Figures 17 to 21, showing the average and the per-cluster BPB (Math, Code, QA) as a function of training FLOPs for each configuration.

Per-domain Breakdown. Table 21 extends Table 5 and provides the per-domain BPB breakdown for all ablation configurations evaluated in Section 5.1. These verify that the aggregate loss improvements are consistent across domains.

D.2 Details on the Scaling Law Fits

Data Used for Fitting. Each architecture’s scaling ladder consists of models trained at seven sizes and five Chinchilla multiples per size ($D/N \in \{10, 20, 40, 80, 160\}$), using post-decay checkpoints only (i.e., checkpoints after the learning-rate decay completes). The loss signal is the *average* of 11 held-out validation cross-entropy losses spanning diverse domains (C4, books, Common Crawl, Pes20, Reddit, Stack, Wikipedia, ICE, S2ORC, Pile, and Wikitext-103); this averaging reduces noise from any single domain. Non-embedding parameter counts are used throughout (embedding and language-model head parameters are excluded), and

Table 19 Compute-optimal model size (N^* , billions), dataset size (D^* , billions of tokens), and predicted loss for all three architectures. Each architecture’s fitted $a_{\text{opt}}, b_{\text{opt}}$ determine its compute-optimal allocation ($C = 6ND$). For each loss entry: [loss CI] on the second line; [Δ CI] on the third line (loss reduction relative to the transformer). All CIs are 95% bootstrap intervals.

FLOPs	Transformer			Hybrid GDN			Pure GDN		
	N^*	D^*	Loss	N^*	D^*	Loss	N^*	D^*	Loss
10^{18}	0.2	0.8	3.57	0.2	0.7	3.46	0.2	1.0	3.53
			[3.52, 3.62]			[3.42, 3.49]			[3.50, 3.55]
						[−0.19, −0.05]			[−0.10, +0.01]
10^{19}	0.6	2.9	3.12	0.7	2.3	3.04	0.6	2.9	3.10
			[3.10, 3.14]			[3.02, 3.05]			[3.09, 3.10]
						[−0.11, −0.05]			[−0.05, +0.00]
10^{20}	1.6	10.2	2.78	2.2	7.4	2.71	2.0	8.2	2.76
			[2.75, 2.79]			[2.69, 2.72]			[2.74, 2.77]
						[−0.10, −0.04]			[−0.04, +0.01]
10^{21}	4.7	35.5	2.51	7.0	24.0	2.46	7.3	22.9	2.49
			[2.45, 2.55]			[2.41, 2.48]			[2.46, 2.52]
						[−0.11, +0.01]			[−0.07, +0.04]
10^{22}	13.5	123.6	2.31	21.6	77.2	2.27	26.0	64.1	2.27
			[2.21, 2.36]			[2.18, 2.30]			[2.23, 2.33]
						[−0.14, +0.06]			[−0.10, +0.08]
10^{23}	38.7	430.2	2.15	67.0	248.7	2.12	93.0	179.2	2.10
			[2.00, 2.23]			[2.00, 2.16]			[2.03, 2.18]
						[−0.17, +0.11]			[−0.14, +0.12]

Table 20 Projected token requirements across model scales (target loss = 2.474, selected as the minimum of all observed training losses). Savings $> 1\times$ means fewer tokens needed than the transformer. 95% bootstrap CI shown below each estimate. See also Figure 10(b).

	Transformer		Hybrid		Pure GDN	
	N	D	D	Savings	D	Savings
1B		776.5B	587.1B	1.32 \times	949.6B	0.82 \times
				[0.67, 4.02]		[0.45, 2.50]
3B		90.0B	57.2B	1.57 \times	79.5B	1.13 \times
				[1.11, 2.50]		[0.77, 1.63]
7B		35.1B	20.9B	1.68 \times	27.0B	1.30 \times
				[0.91, 3.47]		[0.68, 2.30]
13B		21.5B	12.3B	1.75 \times	15.2B	1.41 \times
				[0.84, 4.12]		[0.65, 2.75]
30B		13.1B	7.2B	1.82 \times	8.4B	1.56 \times
				[0.77, 4.98]		[0.61, 3.44]
70B		9.0B	4.8B	1.89 \times	5.3B	1.70 \times
				[0.71, 5.81]		[0.57, 4.23]

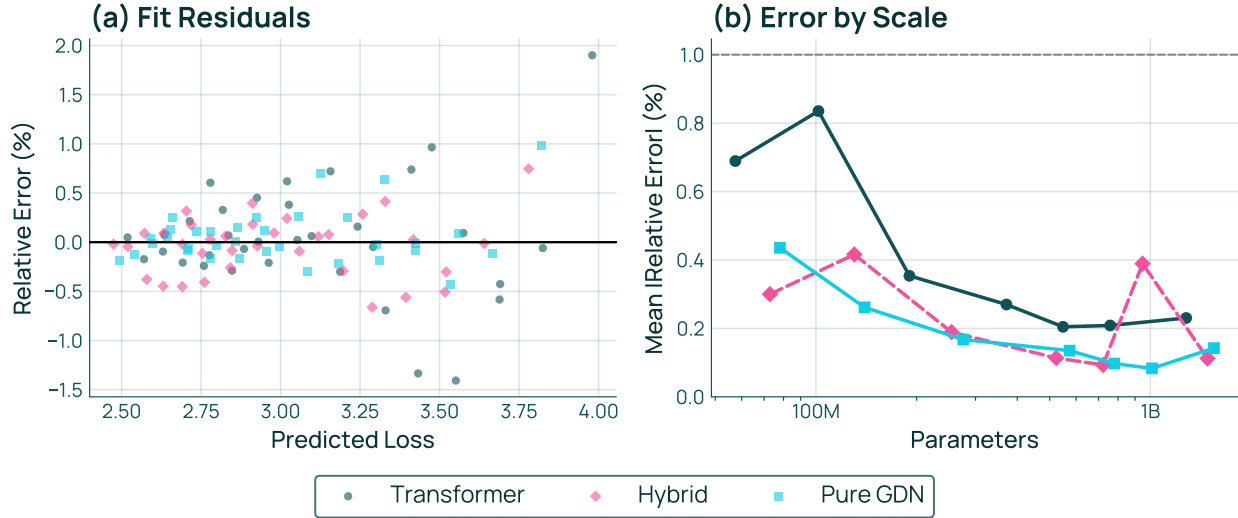


Figure 14 Scaling law fit diagnostics. **(a)** Residuals are small and unbiased for all architectures. **(b)** Mean absolute relative error stays below 1% across all model sizes, confirming reliable fits. Overall: Olmo 3 $R^2 = 0.9976$; Hybrid $R^2 = 0.9993$; Pure GDN $R^2 = 0.9994$.

architecture-specific FLOPs are computed analytically using the parallel (chunkwise) formulations described in Section D.4.

Scaling Law Fitting. We fit the Chinchilla parametric form $L(N, D) = E + A/N^\alpha + B/D^\beta$ by minimizing a Huber loss over the log-residuals, using a multi-start grid optimization (with six slices of the parameter space) to avoid local minima. Bootstrap confidence intervals (95%, $n = 1,000$ resamples) are computed by resampling data points with replacement and re-fitting.

Construction of Figure 8. In each panel, thin curves connect the five post-decay checkpoints (one per Chinchilla multiple) for a given model size, and a bold curve shows the fitted scaling law $L(N, D)$ evaluated at the *compute-optimal frontier*: for each C , N and D are allocated optimally and the predicted loss is computed. Whereas we normally compute the FLOPs as $C = 3 \times F_{\text{fwd}} \times D$, where F_{fwd} is the architecture-specific forward-pass FLOPs per token (see Section D.4), in Figure 8(a) we use the simplification $C = 6ND$ for all architectures. In panel (b), thin curves connect checkpoints of different model sizes at the same Chinchilla multiple, and in panel (c), they connect checkpoints of the same size at different Chinchilla factors. In both cases, the bold curve is evaluated at the largest Chinchilla multiple ($D/N = 160$).

Construction of Figure 10. Both panels are derived from the fitted scaling laws by analytically inverting the loss function. Panel (a) solves $L(N, D) = L_{\text{target}}$ for D as a function of N :

$$D(N) = \left(\frac{B}{L_{\text{target}} - E - A/N^\alpha} \right)^{1/\beta},$$

where L_{target} is set to the minimum observed training loss. This gives the projected number of training tokens each architecture needs to reach that target as a function of model size. Panel (b) plots the savings factor, defined as the ratio of the transformer’s token requirement to each other architecture’s token requirement at matched model size, i.e., $D_{\text{Transformer}}(N)/D_{\text{arch}}(N)$.

Construction of Table 19. Here, we again use the simplification $C = 6ND$. For each compute budget $C = 6ND$, the compute-optimal allocation is derived from the first-order condition $\alpha A/N^\alpha = \beta B/D^\beta$, which

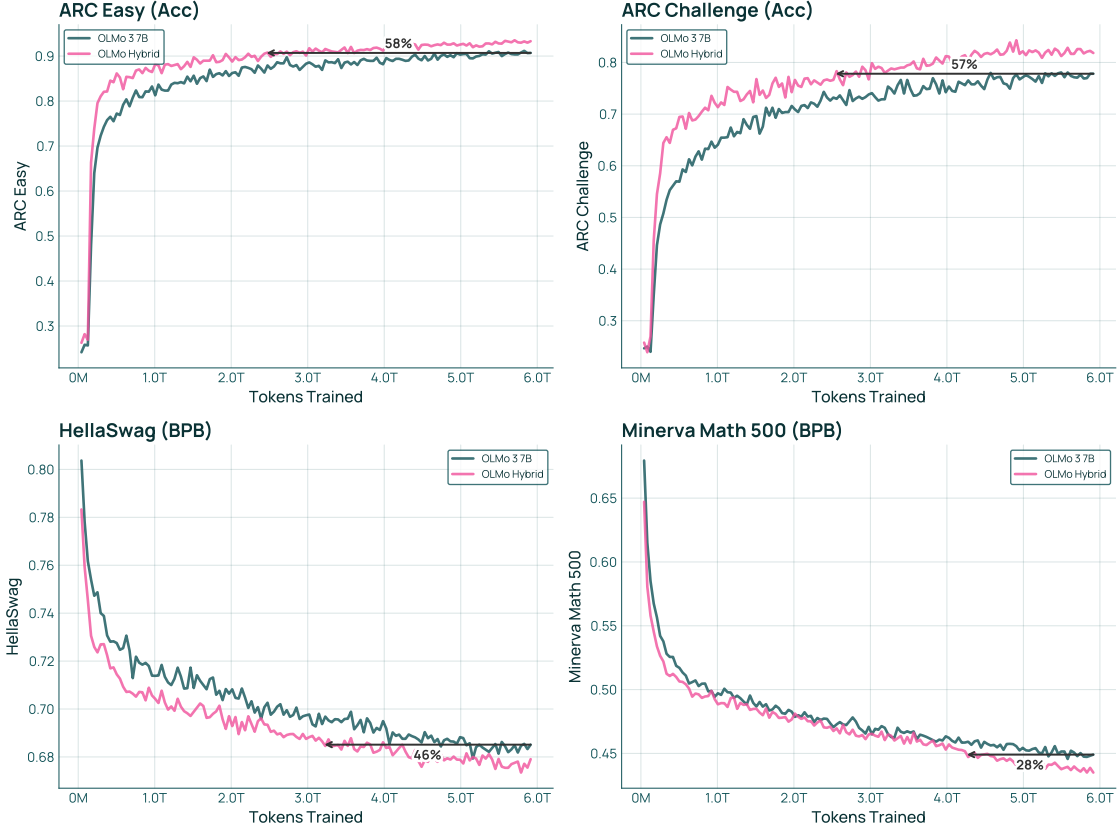


Figure 15 Extended downstream eval training curves comparing Olmo Hybrid 7B and Olmo 3 7B across 6 benchmarks, sorted by token efficiency (see Figure 1 for CE loss and MMLU training curves). Olmo Hybrid reaches Olmo 3’s final performance using 19–58% fewer tokens depending on the benchmark.

gives

$$N^* = \left(\frac{C}{6G} \right)^{a_{\text{opt}}}, \quad D^* = \frac{C}{6N^*},$$

where $G = (\beta B / \alpha A)^{1/\beta}$ and $a_{\text{opt}} = \beta / (\alpha + \beta)$ is each architecture’s fitted compute-optimal exponent. The predicted loss $L(N^*, D^*)$ and the difference Δ relative to the transformer are reported with 95% bootstrap confidence intervals.

D.3 Architectural Details

All models in our scaling ladder and ablation experiments share the same base Olmo 3 hyperparameters at each size (see Table 22): model dimension d , number of attention heads h , number of layers l , and vocabulary size $V = 100,352$ (the Dolma 2 tokenizer, padded for efficient embedding lookups). We provide some additional details on the architecture below.

Olmo 3 (Transformer Baseline). Each Olmo 3 layer consists of a multi-head self-attention sub-layer followed by a SwiGLU MLP (Shazeer, 2020), with RMSNorm (Zhang and Sennrich, 2019) applied before each sub-layer (pre-norm). Attention uses rotary position embeddings (RoPE; Su et al., 2024) and QK-norm. No grouped-query attention is used: the number of key-value heads equals the number of query heads. The MLP hidden dimension is $\lfloor \frac{3}{2} \cdot \frac{8d}{3} \rfloor_{256}$, where $\lceil \cdot \rceil_{256}$ rounds up to the nearest multiple of 256. The embedding and language-model head matrices are untied.

Hybrid (GDN-Transformer). Hybrid GDN models use the same d , h , and l as Olmo 3 but replace a fraction of the attention sub-layers with GDN sub-layers (Yang et al., 2025a). In our default configuration, every r -th

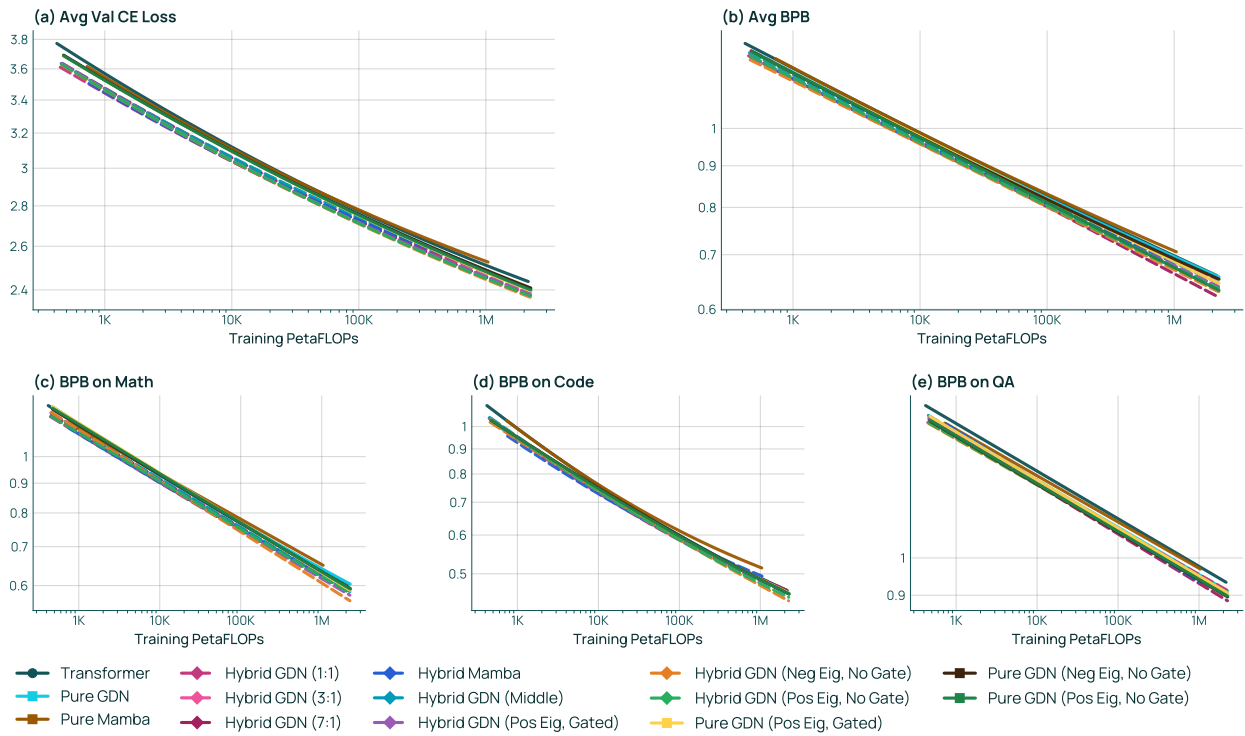


Figure 16 Evaluation metrics vs. training FLOPs. **(a)** Average validation cross-entropy loss across 11 held-out corpora (C4, Dolma Books/CC/pes2o/Reddit/Stack/Wiki, ICE, M2D2 S2ORC, Pile, Wikitext-103; lower is better); this is the same loss used to fit the Chinchilla scaling laws. **(b)** Base Easy Suite average BPB (lower is better). **(c)-(e)** Per-cluster Base Easy BPB for Math, Code, and QA respectively. The thick line is the fitted compute-optimal frontier.

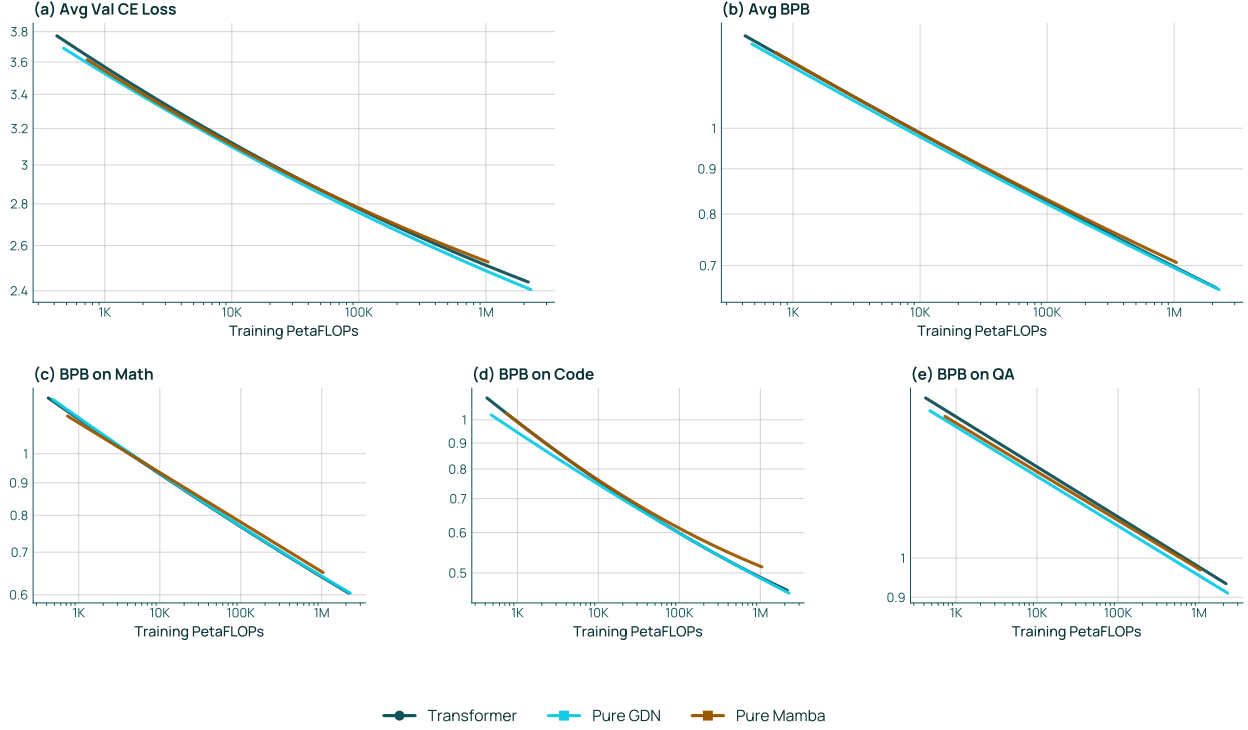


Figure 17 Evaluation metrics vs. training FLOPs for *pure* architectures (Transformer, pure GDN, pure Mamba2). Panels follow the same layout as Figure 16: (a) average validation cross-entropy loss, (b) Base Easy Suite average BPB, and (c)–(e) per-cluster BPB for Math, Code, and QA. The thick line is the fitted compute-optimal frontier.

layer is an attention layer and the remaining layers are GDN layers, where r is the *transformer ratio* (default $r=4$, i.e., a 3:1 linear layer to attention ratio). We additionally enforce the final layer be an attention layer; if it is not already selected by the every- r th rule, it is added.

Each GDN sub-layer computes query, key, and value projections with head dimension $h_{\text{GDN}} = \lceil 0.75 \cdot d/h \rceil_{128}$, yielding a key dimension of $k = h \cdot h_{\text{GDN}}$ and value dimension $v = h \cdot 2h_{\text{GDN}}$ (i.e., the value dimension is expanded by a factor of 2). The GDN sub-layer further includes two scalar per-head parameters (A_{\log} and dt_{bias}), short depthwise convolutions (kernel size 4) over the q , k , and v streams, a gate projection, and an output projection. Each GDN layer retains the same SwiGLU MLP and two RMSNorms as in the Olmo 3 block.

Pure GDN. Pure GDN models replace *all* attention layers with GDN layers (i.e., $r=0$) and do not force a final attention layer. All other hyperparameters are identical to the hybrid variant.

Hybrid Mamba2 (Mamba2-Transformer). Hybrid Mamba2 models follow the same layer interleaving scheme as Hybrid GDN ($r=4$ by default with a forced final attention layer), but substitute Mamba2 (Dao and Gu, 2024) for the non-attention layers. The Mamba2 sub-layer uses an expansion factor of 2 (intermediate size = $2d$), state size $n = 128$, $n_{\text{groups}} = 1$, and a depthwise convolution with kernel size 4. In our hybrid Mamba2 configuration, the Mamba2 layers retain the full MLP from the attention blocks.

D.4 Parameter Count and FLOP Computations

We report total (non-embedding) parameter counts and forward-pass floating-point operations (FLOPs) per token for each model. Both quantities are computed analytically from the architecture specification; the formulas are detailed below.

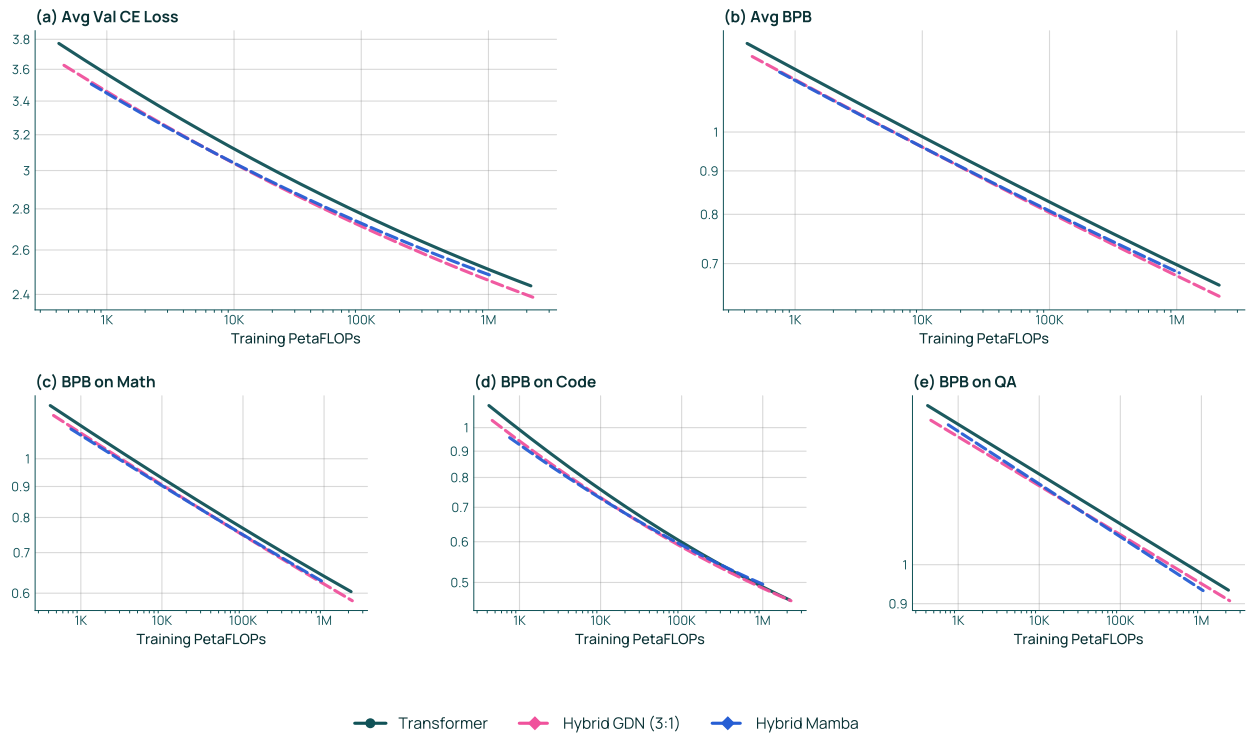


Figure 18 Evaluation metrics vs. training FLOPs for *hybrid* architectures (GDN hybrid with interleaved and middle placement, Mamba2 hybrid). Panels follow the same layout as Figure 16: (a) average validation cross-entropy loss, (b) Base Easy Suite average BPB, and (c)–(e) per-cluster BPB for Math, Code, and QA. The thick line is the fitted compute-optimal frontier.

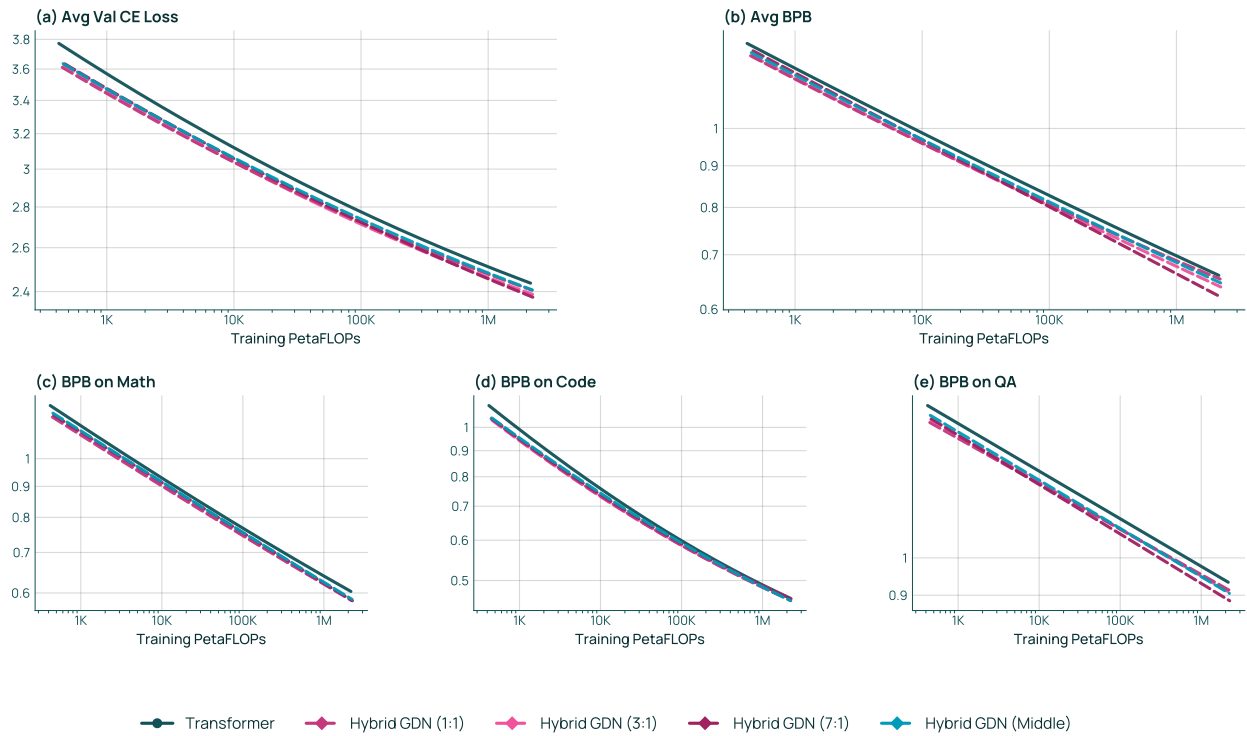


Figure 19 Evaluation metrics vs. training FLOPs for GDN hybrid models with varying linear-to-attention ratios (1:1, 3:1, 7:1). Panels follow the same layout as Figure 16: (a) average validation cross-entropy loss, (b) Base Easy Suite average BPB, and (c)–(e) per-cluster BPB for Math, Code, and QA. The thick line is the fitted compute-optimal frontier.

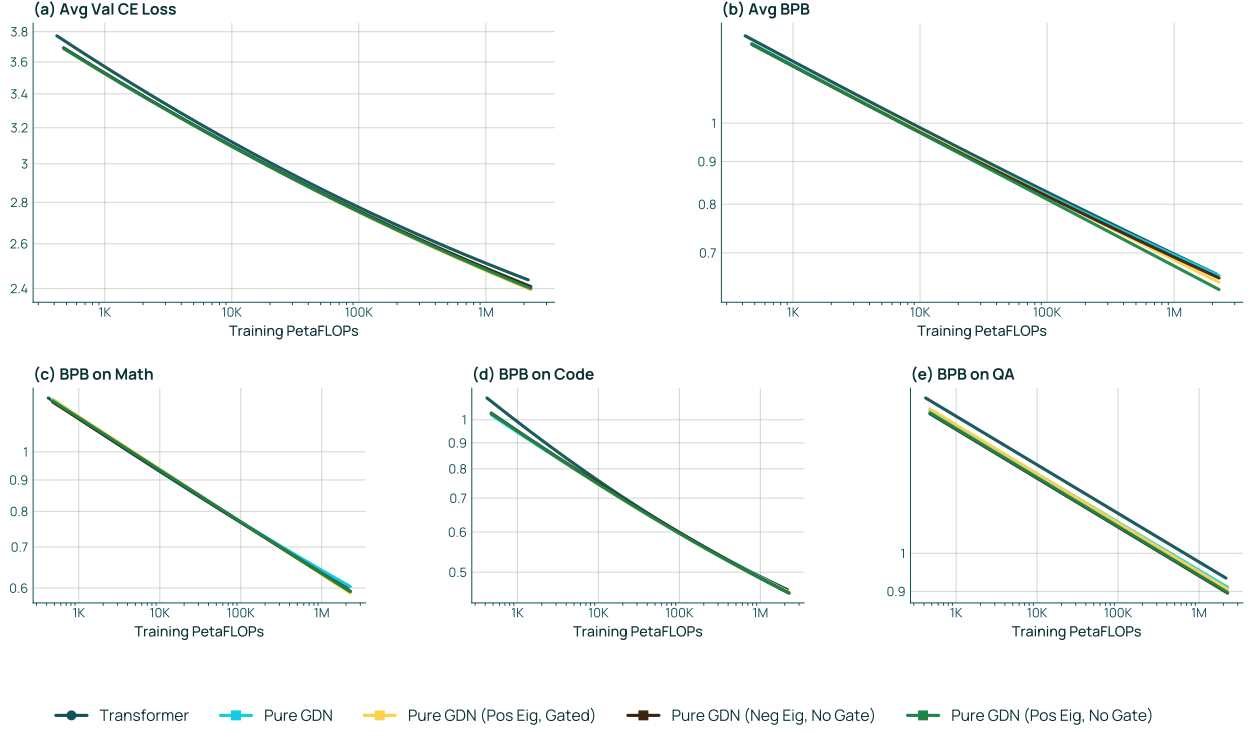


Figure 20 Evaluation metrics vs. training FLOPs for GDN ablations (e.g., GDN with and without negative eigenvalues). Panels follow the same layout as Figure 16: (a) average validation cross-entropy loss, (b) Base Easy Suite average BPB, and (c)–(e) per-cluster BPB for Math, Code, and QA. The thick line is the fitted compute-optimal frontier.

Parameter Counting. The total parameter count is the sum of embedding, language-model head, per-layer, and final layer-norm parameters:

$$N_{\text{total}} = \underbrace{V \cdot d}_{\text{embed}} + \underbrace{V \cdot d}_{\text{LM head}} + \sum_{\ell=1}^l N_{\text{layer}}^{(\ell)} + \underbrace{d}_{\text{final norm}}, \quad (2)$$

where $N_{\text{layer}}^{(\ell)}$ depends on the layer type. Non-embedding parameters are $N_{\text{non-emb}} = N_{\text{total}} - V \cdot d$.

For an **attention layer**, the sub-layer parameter count is:

$$N_{\text{attn}} = \underbrace{4d^2}_{\text{Q,K,V,O projections}} + \underbrace{2d}_{\text{QK-norm}} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{SwiGLU MLP}} + \underbrace{2d}_{\text{2 RMSNorms}}, \quad (3)$$

where $d_{\text{MLP}} = \lceil \frac{3}{2} \cdot \frac{8d}{3} \rceil_{256}$ is the MLP hidden size.

For a **GDN layer**, letting $k = h \cdot h_{\text{GDN}}$ and $v = 2k$ denote the total key and value dimensions:

$$N_{\text{GDN}} = \underbrace{d(2k + v + 2h)}_{\text{q,k,v,a,b projections}} + \underbrace{2h}_{A_{\log}, dt_{\text{bias}}} + \underbrace{(2k + v) \cdot 4}_{\text{short convolutions}} + \underbrace{dv}_{\text{gate projection}} + \underbrace{v}_{\text{norm}} + \underbrace{vd}_{\text{output projection}} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{MLP}} + \underbrace{2d}_{\text{norms}}. \quad (4)$$

For a **Mamba2 layer**, letting $e = 2d$ (intermediate size), $c = e + 2n_{\text{groups}} \cdot n$ (convolution dimension), and $p = e + c + h$ (projection size):

$$N_{\text{mamba2}} = \underbrace{d \cdot p + p}_{\text{in projection}} + \underbrace{c \cdot 4}_{\text{conv1d}} + \underbrace{3h}_{\text{dt,A,D}} + \underbrace{e}_{\text{norm}} + \underbrace{e \cdot d + d}_{\text{out projection}} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{SwiGLU MLP}} + \underbrace{2d}_{\text{layer norms}}. \quad (5)$$

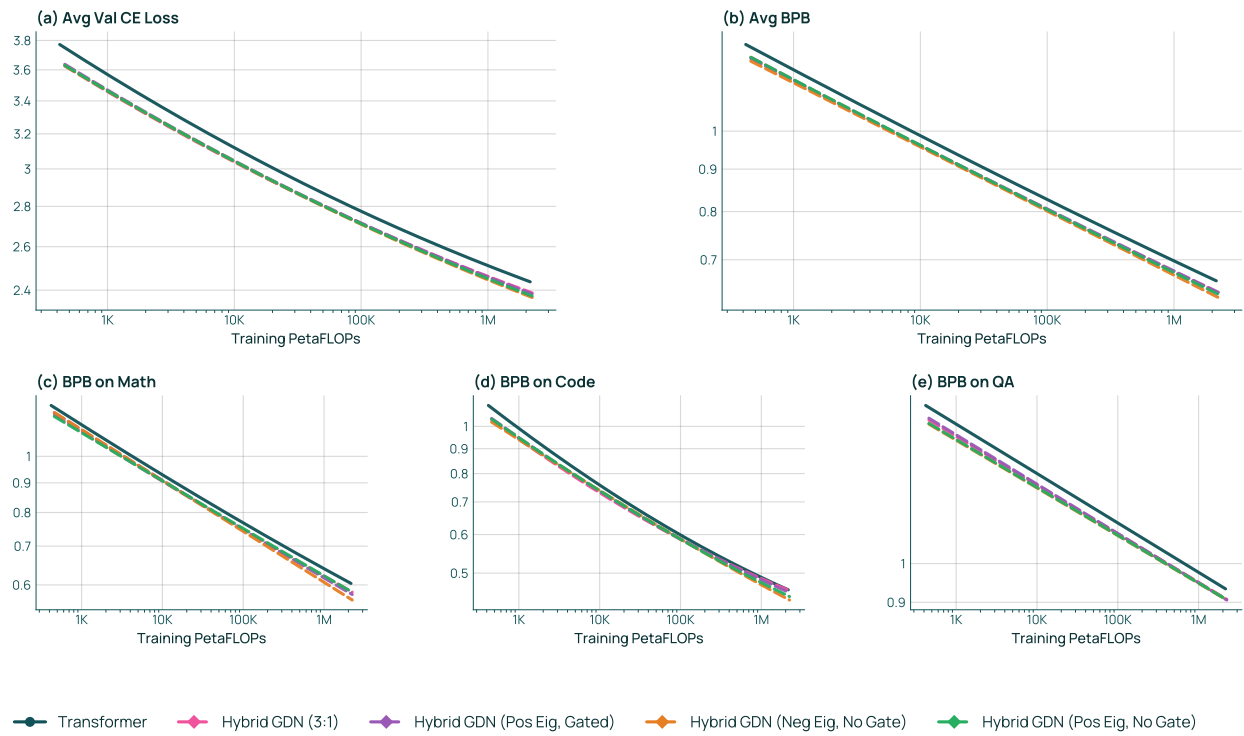


Figure 21 Evaluation metrics vs. training FLOPs for GDN hybrid ablations (e.g., hybrid GDN with and without negative eigenvalues). Panels follow the same layout as Figure 16: (a) average validation cross-entropy loss, (b) Base Easy Suite average BPB, and (c)–(e) per-cluster BPB for Math, Code, and QA. The thick line is the fitted compute-optimal frontier.

D.4.1 FLOP Counting

We estimate forward-pass FLOPs per token using the standard convention $\text{FLOPs} = 2 \times \text{MACs}$ (multiply-accumulate operations). The total FLOPs per token are:

$$F = 2 \left(\underbrace{d \cdot V}_{\text{LM head}} + \sum_{\ell=1}^l M_{\text{layer}}^{(\ell)} \right), \quad (6)$$

where $M_{\text{layer}}^{(\ell)}$ is the per-token MAC count for layer ℓ . Embedding lookups are excluded as they involve no arithmetic. For **attention layers**, the MACs per token include both the projections and the sequence-length-dependent attention computation. For causal attention, we use the average context length $s_{\text{eff}} = s/2$:

$$M_{\text{attn}} = \underbrace{4d^2}_{\text{Q,K,V,O}} + \underbrace{h \cdot \frac{d}{h} \cdot s_{\text{eff}}}_{\text{scores (QK}^\top)} + \underbrace{h \cdot \frac{d}{h} \cdot s_{\text{eff}}}_{\text{output (attn} \cdot V)} + \underbrace{\frac{5}{2} \cdot h \cdot s_{\text{eff}}}_{\text{softmax}} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{MLP}} = 4d^2 + ds + 3d \cdot d_{\text{MLP}}, \quad (7)$$

where s is the sequence length. The factor $\frac{5}{2}$ in front of the softmax contribution accounts for the $5s_{\text{eff}}$ FLOPs usually attributed to softmax normalization (Beck et al., 2026) divided by 2 since the expression computes MACs.

For the linear recurrent layers (GDN and Mamba2), we first present per-token MACs in the recurrent formulation, and then describe the chunkwise parallel formulation used during training.

Recurrent Formulation. For **GDN layers**, the per-token MACs include projections, convolutions, the recurrence, and the MLP. The delta rule requires three state interactions (retention projection, state update, output projection):

$$M_{\text{GDN}} = \underbrace{d(2k + v + 2h)}_{\text{linear projs}} + \underbrace{4(2k + v)}_{\text{depthwise convs}} + \underbrace{dv}_{\text{gate proj}} + \underbrace{vd}_{\text{out proj}} + \underbrace{3 \cdot h \cdot h_{\text{GDN}} \cdot 2h_{\text{GDN}}}_{\text{recurrence (Sk, uv}^\top, Sq)} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{MLP}}. \quad (8)$$

For **Mamba2 layers**, the recurrence involves two state interactions (state update and output projection):

$$M_{\text{mamba2}} = \underbrace{d \cdot p}_{\text{in proj}} + \underbrace{4c}_{\text{conv1d}} + \underbrace{e \cdot d}_{\text{out proj}} + \underbrace{2 \cdot h \cdot \frac{e}{h} \cdot n}_{\text{SSM recurrence (vk}^\top, Sq)} + \underbrace{3d \cdot d_{\text{MLP}}}_{\text{MLP}}. \quad (9)$$

Parallel (Chunkwise) Formulation. While the recurrent formulation above describes inference-time FLOPs, training utilizes hardware-efficient chunkwise parallel algorithms. For Gated DeltaNet, we utilize the **Extended WY Representation** algorithm (Yang et al., 2025a), which folds the data-dependent decay terms directly into the update matrices, avoiding the need for log-space computations or sub-tiling required by previous gated linear attention methods (Yang et al., 2024b). Training FLOPs account for the overhead of intra-chunk local attention and inter-chunk state passing.

For **GDN layers**, the Extended WY algorithm introduces overheads proportional to the chunk size L_{chunk} (set to 256) for constructing the kernel, update matrices (W, U), and computing local attention.

$$M_{\text{GDN,train}} = \underbrace{M_{\text{proj,conv,MLP}}}_{\text{sequence-independent}} + \underbrace{L_{\text{chunk}}(3k + 2v)}_{\text{intra-chunk overhead (Kernel, W/U, Attn)}} + \underbrace{3 \cdot \frac{kv}{h}}_{\text{inter-chunk state passing}}, \quad (10)$$

where $M_{\text{proj,conv,MLP}}$ represents the cost of projections, convolutions, and MLPs (identical to the recurrent formulation). The intra-chunk coefficient $(3k + 2v)$ accounts for the construction of KK^\top , W , U , QK^\top , and the final output computation. The inter-chunk cost $(3kv/h)$ represents the three matrix multiplications required to propagate the hidden state (QS^\top , $U^\top K$, WS^\top).

For **Mamba2 layers**, utilizing the sequential chunkwise algorithm (instead of parallel scan) reduces the state-passing overhead to simple unidirectional updates:

$$M_{\text{mamba2,train}} = \underbrace{M_{\text{proj,conv,MLP}}}_{\text{sequence-independent}} + \underbrace{2 \cdot L_{\text{chunk}} \cdot e}_{\text{intra-chunk SSD mixing}} + \underbrace{2 \cdot e \cdot n}_{\text{inter-chunk state passing}}, \quad (11)$$

where e is the intermediate size and n is the state size. The inter-chunk cost accounts for the unidirectional state passing (state update $S + VK^\top$ and output computation QS^\top) required by the sequential algorithm.

For the scaling law fits, we use the analytically computed training FLOPs per token F (using the parallel formulation) to obtain the compute estimate $C = 3FD$ (accounting for forward and backward passes).

E Proofs: Increased Expressive Power Improves Scaling

In the main text, we have theoretically explained how increasing an architecture’s expressivity can lead to better scaling on a language modeling objective, working in the idealized setup of the quantization model for neural scaling laws. We now clarify details of the quantization model and provide deferred proofs.

Michaud et al. (2023) introduce the quantization model of neural scaling laws, which we will augment to take into account expressivity limitations of different architectures. First, we fully define the quantization model, starting with the notion of a learnable task:

Definition 3: Learnable Task

Task k is learnable with D tokens if $D \geq \frac{1}{p_k} T_k$, i.e., we can expect to observe enough relevant tokens leveraging task k within the D tokens.

In the quantization model, an idealized LM learner proceeds by allocating parameters to learn learnable tasks, ordered by their frequency:

Definition 4: Quantization Model

The learner is given N parameters and D samples. It proceeds by ranking all tasks k learnable with D samples by their frequency p_k in the data, greedily spending C_k samples to learn task k in this order until the parameter budget N is exceeded.

Let X_k be a random variable representing the loss achieved on task k . We will analyze \tilde{L} , the expected loss across all tasks, which can be defined as

$$\tilde{L} = \mathbb{E} \left[\sum_{k=1}^{\infty} X_k p_k \right].$$

We incorporate expressivity into the quantization model via the following assumptions, restated from the main text:

Assumption 1: Only Some Tasks Are Expressible

For a given architecture, each task is either expressible or inexpressible, modeled as an *iid* boolean random variable where the probability that any individual task is expressible is $1 - \epsilon$.

Assumption 2: Expressible Tasks Can Be Learned to Lower Loss

Tokens leveraging unlearned tasks incur loss L_0 . If a task k has been learned, the loss incurred by the learner on a token leveraging k is reduced to $L_0 - \Delta$ if k is expressible by that learner and to $L_0 - \Delta'$ if k is inexpressible, for some $\Delta, \Delta' > 0$ and $\Delta' \leq \Delta$.

Assumption 3: Expressible Tasks Can Be Learned with Fewer Parameters and Tokens

Each expressible task is representable with C parameters and learnable with T relevant tokens. An inexpressible task requires $C' \geq C$ parameters to represent approximately and is learnable with $T' \geq T$ relevant tokens.

It follows from these assumptions that, for an unlearned task, $X_k = b$, but, for a learned task,

$$X_k = \begin{cases} a & \text{w.p. } 1 - \epsilon \\ a' & \text{otherwise.} \end{cases}$$

In contrast, for any task k , we have the following sample complexity and parameter requirements:

$$T_k = \begin{cases} T & \text{w.p. } 1 - \epsilon \\ T' & \text{otherwise.} \end{cases}$$

$$C_k = \begin{cases} C & \text{w.p. } 1 - \epsilon \\ C' & \text{otherwise.} \end{cases}$$

We will assume $T' \geq T$, that is, inexpressible tasks require at least as many samples to approximate as expressible tasks. It is natural to imagine $C' \geq C$, though we may also consider the case where $C' = 0$ and $\Delta' = 0$, i.e., inexpressible tasks are completely ignored by the learner.

E.1 Scaling Law with Tasks Learned

We first slightly generalize the original scaling law from Michaud et al. (2023) in the following way:

Lemma 3 *For every learned task k , let X_k be an independent random variable with mean a . Similarly, for every unlearned task k , let X_k be an independent random variable with mean b . Then the expected loss \tilde{L}_n after learning n tasks is*

$$\tilde{L}_n = a + \frac{b - a}{\zeta(\alpha + 1)} \sum_{k=n+1}^{\infty} k^{-(\alpha+1)}.$$

Further, the following L_n closely approximates \tilde{L}_n :

$$L_n = a + \frac{b - a}{\alpha \zeta(\alpha + 1)} n^{-\alpha},$$

in the sense that $L_{n+1} \leq \tilde{L}_n \leq L_n$, and thus the true loss \tilde{L}_n rapidly converges to the power law L_n .

Proof. The expression for the first part, \tilde{L}_n , is a straightforward extension to the original derivation from Michaud et al. (2023). For the approximation, we observe that the terms in the summation $\sum_{n+1}^{\infty} n^{-(\alpha+1)}$ represent a step function, which is naturally lower and upper bounded by the corresponding continuous “envelopes” of the step function as follows:

$$\int_{n+1}^{\infty} k^{-(\alpha+1)} dk \leq \sum_{k=n+1}^{\infty} k^{-(\alpha+1)} \leq \int_n^{\infty} k^{-(\alpha+1)} dk$$

which simplifies to

$$\frac{(n+1)^{-\alpha}}{\alpha} \leq \sum_{k=n+1}^{\infty} k^{-(\alpha+1)} \leq \frac{n^{-\alpha}}{\alpha}.$$

It follows that \tilde{L}_n is sandwiched between L_{n+1} and L_n , and rapidly approaches L_n in the following sense: [AS: @will: added this for concreteness; have a look when you get a chance; could revisit making the

claim in the lemma statement of relative error being only $O(1/n)$.]

$$\begin{aligned}
\frac{L_n - \tilde{L}_n}{\tilde{L}_n} &= \frac{L_n}{\tilde{L}_n} - 1 \leq \frac{L_n}{L_{n+1}} - 1 \\
&= \frac{a + \frac{b-a}{\alpha \zeta(\alpha+1)} n^{-\alpha}}{a + \frac{b-a}{\alpha \zeta(\alpha+1)} (n+1)^{-\alpha}} - 1 \\
&\leq \frac{n^{-\alpha}}{(n+1)^{-\alpha}} - 1 \\
&= \frac{(n+1)^\alpha - n^\alpha}{n^\alpha} \\
&= \frac{\alpha n^{\alpha-1} + \frac{\alpha(\alpha-1)}{2!} n^{\alpha-2} + \dots}{n^\alpha} && \text{by Taylor series expansion} \\
&= O(1/n)
\end{aligned}$$

Thus, the relative error $(L_n - \tilde{L}_n)/\tilde{L}_n$ is only $O(1/n)$. \square

In the upcoming results, we will follow Michaud et al. (2023) in using the approximation L_n from Lemma 3. Having established a general result about analyzing the quantization model, we consider its instantiation in the expressivity-aware case:

Corollary 3.1 *Adopt Assumptions 1 and 2, i.e., tasks are expressible w.p. $1 - \epsilon$, achieving loss $L_0 - \Delta$ when learned, or inexpressible w.p. ϵ , achieving loss $L_0 - \Delta'$ when learned. Then the expected loss \tilde{L}_n^ϵ after learning n tasks is closely approximated by L_n^ϵ where:*

$$L_n^\epsilon = L_\infty^\epsilon + \frac{L_0 - L_\infty^\epsilon}{\alpha \zeta(\alpha + 1)} \cdot n^{-\alpha},$$

and $L_\infty^\epsilon = L_0 - (1 - \epsilon)\Delta - \epsilon\Delta'$.

The irreducible loss term L_∞^ϵ will be the same in this scaling law as in the upcoming scaling laws for parameters and tokens. Quite naturally, it interpolates between the reduced loss of expressible and inexpressible tasks. Assuming $\Delta' < \Delta$, then, for any $\tilde{\epsilon} > \epsilon$, we will have $L_\infty^{\tilde{\epsilon}} > L_\infty^\epsilon$. Furthermore, the following shows that this reduction in irreducible loss translates to a reduction in loss everywhere along the loss curve:

Lemma 4 *Fix $f, f' : \mathbb{N} \rightarrow (0, 1)$ with $f'(n) \geq f(n)$ for all n . Define*

$$\begin{aligned}
L_n &= a + (b - a)f(n) \\
L'_n &= a' + (b - a')f'(n).
\end{aligned}$$

Then, if $a' > a$, it holds that $L'_n > L_n$ for all n .

Proof. Define $\Delta L_n = L'_n - L_n$. We show that, if $a' > a$, it holds that $\Delta L_n > 0$ for all n . By definition,

$$\begin{aligned}
\Delta L_n &= a' - a + (b - a')f'(n) - (b - a)f(n) \\
&\geq a' - a + (b - a')f(n) - (b - a)f(n) \\
&= (a' - a)(1 - f(n)).
\end{aligned}$$

Since $f(n) \in (0, 1)$, we have $\Delta L_n > 0$ as long as $a' > a$. \square

Combining Corollary 3.1 and Lemma 4, the fact that we have $L_\infty^{\tilde{\epsilon}} > L_\infty^\epsilon$ implies that $L_n^{\tilde{\epsilon}} > L_n^\epsilon$ for any n .

E.2 Parameter Scaling Law

Define $\tilde{L}(N) = \mathbb{E}[L_{n_N}]$, where n_N is the expected number of tasks we can learn with N parameters.

Lemma 5 *Adopt the assumptions of Lemma 3. Further, assume the parameter cost of task k is a random variable C_k , independent across k with mean c . Then, as a function of the number of parameters N , the loss $\tilde{L}(N)$ in the quantization model is closely approximated by a power law, i.e., $\tilde{L}(N) \approx L(N)$ where:*

$$L(N) = a + c^\alpha \cdot \frac{b - a}{\alpha \zeta(\alpha + 1)} \cdot N^{-\alpha}.$$

Proof. The expected number of tasks n_N we can learn with N parameters satisfies

$$N \geq \sum_{k=1}^{n_N} C_k = n_N \cdot c,$$

which implies $n_N = \lfloor N/c \rfloor \approx N/c$. Plugging this into the expression for L_n from Lemma 3 yields the following approximation for the expected loss:

$$\begin{aligned} L(N) &= a + \frac{b - a}{\alpha \zeta(\alpha + 1)} (N/c)^{-\alpha} \\ &= a + c^\alpha \cdot \frac{b - a}{\alpha \zeta(\alpha + 1)} \cdot N^{-\alpha}. \end{aligned} \quad \square$$

We now consider the form of $L(N)$ under the assumptions about expressivity awareness. By Assumptions 1 and 3, we have $C_n = C$ with probability $1 - \epsilon$ and $C_n = C'$ with probability ϵ . Thus, we obtain:

Corollary 5.1 *Adopt Assumptions 1 to 3, i.e., tasks are expressible w.p. $1 - \epsilon$, requiring C parameters or inexpressible w.p. ϵ , requiring C' parameters. Then, as a function of the number of parameters N , the loss $\tilde{L}(N)$ in the quantization model is closely approximated by a power law, i.e., $\tilde{L}(N) \approx L(N)$ where:*

$$L(N) = L_\infty^\epsilon + A_\epsilon \cdot \frac{L_0 - L_\infty^\epsilon}{\alpha \zeta(\alpha + 1)} \cdot N^{-\alpha},$$

where L_∞^ϵ and L_0 are defined as in Corollary 3.1 and

$$A_\epsilon = (C + \epsilon(C' - C))^\alpha.$$

The cost C' does not change the irreducible loss, though it does impact the slope of the scaling trend. In particular, with $C' = C$, we obtain the coefficient $c^\alpha = C^\alpha$, and with $C' = 0$, we recover the coefficient $c^\alpha = (1 - \epsilon)^\alpha C^\alpha$. Further, under any nontrivial instantiation of the model (i.e., where inexpressible tasks either reduce loss less or cost more), increasing expressivity will decrease $L(N)$ for all N :

Corollary 5.2 *Assume $C' \geq C$. If $\Delta' < \Delta$ or $C' > C$, then $L(N)$ strictly decreases as ϵ decreases, elementwise for all N .*

Proof. If $\Delta' = \Delta$ but $C' > C$, then $L_\infty^\epsilon = L_0 - \Delta$ independent of ϵ , but A_ϵ is increasing with ϵ . Thus, $L(N)$ will increase with ϵ for all N .

If we have $\Delta' < \Delta$, then L_∞^ϵ is increasing with ϵ . It also holds that, for $\epsilon' > \epsilon$, $A_{\epsilon'} \geq A_\epsilon$. We apply Lemma 4 to conclude that $L(N)$ is increasing with ϵ elementwise for all N . \square

E.3 Data Scaling Law

We first generalize Lemma 3 in the following way:

Lemma 6 Let n_0, \dots, n_m be a finite sequence of phases, with $n_0 = 0$ and $n_m = \infty$. For each $1 \leq \ell \leq m$, assume that, for all k such that $n_{\ell-1} < k \leq n_\ell$, all X_k are independent with mean $\mu_{\ell-1}$. Then the expected loss \tilde{L} across all tasks is closely approximated by a power law, i.e., $\tilde{L} \approx L$ where:

$$L = \mu_0 + \frac{1}{\alpha \zeta(\alpha + 1)} \sum_{\ell=1}^{m-1} (\mu_\ell - \mu_{\ell-1}) \cdot n_\ell^{-\alpha}.$$

Proof. We can first write the loss as a weighted sum over losses coming from different phases:

$$\begin{aligned} \tilde{L} &= \mathbb{E} \left[\sum_{k=1}^{\infty} X_k p_k \right] \\ &= \mathbb{E} \left[\sum_{\ell=1}^m \sum_{k=n_{\ell-1}+1}^{n_\ell} X_k p_k \right] \\ &= \sum_{\ell=1}^m \mu_{\ell-1} \sum_{k=n_{\ell-1}+1}^{n_\ell} p_k. \end{aligned}$$

At this point, for each series $\ell < m$, we add the missing terms $\sum_{k=n_{\ell+1}}^{\infty} p_k$ to the series and subtract it from the remaining series. This yields:

$$\begin{aligned} \tilde{L} &= \sum_{\ell=1}^m \mu_{\ell-1} \left(\sum_{k=n_{\ell-1}+1}^{\infty} p_k - \sum_{k=n_\ell+1}^{\infty} p_k \right) \\ &= \left(\sum_{\ell=1}^m \mu_{\ell-1} \sum_{k=n_{\ell-1}+1}^{\infty} p_k \right) - \left(\sum_{\ell=1}^m \mu_{\ell-1} \sum_{k=n_\ell+1}^{\infty} p_k \right) \\ &= \left(\sum_{\ell=1}^m \mu_{\ell-1} \sum_{k=n_{\ell-1}+1}^{\infty} p_k \right) - \left(\sum_{\ell=2}^{m+1} \mu_{\ell-2} \sum_{k=n_{\ell-1}+1}^{\infty} p_k \right) \\ &= \mu_0 \sum_{k=n_0+1}^{\infty} p_k + \left(\sum_{\ell=2}^m (\mu_{\ell-1} - \mu_{\ell-2}) \sum_{k=n_{\ell-1}+1}^{\infty} p_k \right) - \mu_{m-1} \sum_{k=n_m+1}^{\infty} p_k \\ &= \mu_0 \sum_{k=1}^{\infty} p_k + \left(\sum_{\ell=2}^m (\mu_{\ell-1} - \mu_{\ell-2}) \sum_{k=n_{\ell-1}+1}^{\infty} p_k \right) - \mu_{m-1} \sum_{k=\infty}^{\infty} p_k \\ &= \mu_0 + \sum_{\ell=2}^m (\mu_{\ell-1} - \mu_{\ell-2}) \sum_{k=n_{\ell-1}+1}^{\infty} p_k. \end{aligned}$$

At this point, we can simplify the series in a similar way to Michaud et al. (2023) to conclude:

$$\begin{aligned} \tilde{L} \approx L &= \mu_0 + \sum_{\ell=2}^m \frac{\mu_{\ell-1} - \mu_{\ell-2}}{\alpha \zeta(\alpha + 1)} \cdot n_{\ell-1}^{-\alpha} \\ &= \mu_0 + \frac{1}{\alpha \zeta(\alpha + 1)} \sum_{\ell=1}^{m-1} (\mu_\ell - \mu_{\ell-1}) \cdot n_\ell^{-\alpha}. \quad \square \end{aligned}$$

Lemma 7 Adopt Assumptions 1 to 3, i.e., tasks are expressible w.p. $1 - \epsilon$, requiring T relevant tokens to learn, or inexpressible w.p. ϵ , requiring T' relevant tokens to learn. Then, as a function of the token budget D , the expected loss $\tilde{L}(D)$ under the quantization model is closely approximated by a power law, i.e., $\tilde{L}(D) \approx L(D)$ where

$$L(D) = L_\infty^\epsilon + \frac{B_\epsilon}{\alpha \zeta(\alpha + 1)^{1/(\alpha+1)}} \cdot D^{-\alpha/(\alpha+1)},$$

L_∞^ϵ is defined as in Corollary 3.1, and

$$B_\epsilon = (1 - \epsilon)\Delta T^{\alpha/(\alpha+1)} + \epsilon\Delta' T'^{\alpha/(\alpha+1)}.$$

Proof. We consider the tasks that are learned with D tokens. By Definition 3, any expressible task k learnable with D tokens satisfies

$$\begin{aligned} D &\geq \frac{1}{p_k} T \\ &= \zeta(\alpha + 1) k^{\alpha+1} T \end{aligned}$$

Let k_e be the largest index representing a learnable expressible task. It holds that

$$\begin{aligned} k_e &= \left\lfloor \left(\frac{1}{\zeta(\alpha + 1)} \cdot D/T \right)^{1/(\alpha+1)} \right\rfloor \\ &\approx \left(\frac{1}{\zeta(\alpha + 1)} \cdot D/T \right)^{1/(\alpha+1)}. \end{aligned}$$

By the same reasoning and Assumption 1, we have that the maximum index $k_i \leq k_e$ representing an inexpressible task learnable with D tokens satisfies

$$k_i \approx \left(\frac{1}{\zeta(\alpha + 1)} \cdot D/T' \right)^{1/(\alpha+1)}.$$

Now, we will define three phases of tasks, during each of which all task losses are independent and have a different mean: Thus:

1. Phase from $1 \leq k \leq k_i$: all tasks are learned, with expressible tasks obtaining loss $L_0 - \Delta$ and inexpressible tasks obtaining loss $L_0 - \Delta'$. Thus, the expected loss is $L_\infty^\epsilon = L_0 - (1 - \epsilon)\Delta - \epsilon\Delta'$.
2. Phase from $k_i + 1 \leq k \leq k_e$: only expressible tasks are learned obtaining loss Δ . Thus, the expected loss is $L_0 - (1 - \epsilon)\Delta$.
3. Phase from $k_e + 1 \leq k < \infty$: no tasks are learned. Thus, the expected loss is L_0 .

Invoking Lemma 6 with these three phases, the loss $\tilde{L}(D)$ is closely approximated by the following:

$$\begin{aligned} &L_\infty^\epsilon + \frac{1}{\alpha \zeta(\alpha + 1)} \left((1 - \epsilon)\Delta k_e^{-\alpha} + \epsilon\Delta' k_i^{-\alpha} \right) \\ &\approx L_\infty^\epsilon + \frac{(1 - \epsilon)\Delta (D/T)^{-\alpha/(\alpha+1)} + \epsilon\Delta' (D/T')^{-\alpha/(\alpha+1)}}{\alpha \zeta(\alpha + 1)^{1/(\alpha+1)}} \\ &= L_\infty^\epsilon + \frac{(1 - \epsilon)\Delta T^{\alpha/(\alpha+1)} + \epsilon\Delta' T'^{\alpha/(\alpha+1)}}{\alpha \zeta(\alpha + 1)^{1/(\alpha+1)}} \cdot D^{-\alpha/(\alpha+1)} \\ &= L_\infty^\epsilon + \frac{B_\epsilon}{\alpha \zeta(\alpha + 1)^{1/(\alpha+1)}} \cdot D^{-\alpha/(\alpha+1)}. \end{aligned}$$

This final expression is $L(D)$, our desired approximation to the expected loss. \square

The irreducible loss here is the same as that of the task scaling law. The scaling coefficient changes from the expressivity-unaware case by a factor of B_ϵ/B_0 . Moreover, we see that increasing expressivity improves the loss curve:

Corollary 7.1 *If $\Delta' < \Delta$, or $T' > T$, then $L(D)$ strictly decreases as ϵ decreases, elementwise for all D .*

Proof. We will prove this by showing that the derivative of the loss w.r.t. ϵ is strictly positive. Let $Z = \alpha \zeta(\alpha + 1)^{1/(\alpha+1)}$. Recall that the loss $L(D)$ is

$$\begin{aligned} L(D) &= L_\infty^\epsilon + \frac{B_\epsilon}{Z} \cdot D^{-\alpha/(\alpha+1)} \\ &= L_\infty^\epsilon + \frac{(1-\epsilon)\Delta T^{\alpha/(\alpha+1)} + \epsilon\Delta' T'^{\alpha/(\alpha+1)}}{Z} \cdot D^{-\alpha/(\alpha+1)} \\ &= L_0 - (1-\epsilon)\Delta - \epsilon\Delta' + \frac{1}{Z} \cdot \left((1-\epsilon)\Delta \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} + \epsilon\Delta' \left(\frac{T'}{D}\right)^{\alpha/(\alpha+1)} \right). \end{aligned}$$

The derivative of the loss w.r.t. ϵ is therefore

$$\begin{aligned} \frac{\partial}{\partial \epsilon} L(D) &= (\Delta - \Delta') + \frac{1}{Z} \cdot \left(-\Delta \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} + \Delta' \left(\frac{T'}{D}\right)^{\alpha/(\alpha+1)} \right) \\ &\geq (\Delta - \Delta') + \frac{1}{Z} \cdot \left(-\Delta \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} + \Delta' \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} \right) && \text{since } T' \geq T \\ &= (\Delta - \Delta') + \frac{1}{Z} \cdot \left(-(\Delta - \Delta') \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} \right) \\ &= (\Delta - \Delta') \cdot \left(1 - \frac{1}{Z} \cdot \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} \right) \\ &\geq (\Delta - \Delta') \cdot \left(1 - \frac{1}{Z} \right) && \text{since } D \geq T, \end{aligned}$$

which is strictly positive as long as $\Delta' < \Delta$. It follows that, for $\Delta' < \Delta$, $L(D)$ is an increasing function of ϵ , i.e., it increases (pointwise) as ϵ increases and expressivity decreases.

When $\Delta' = \Delta$, the expression for the derivative simplifies to

$$\begin{aligned} \frac{\partial}{\partial \epsilon} L(D) &= \frac{1}{Z} \cdot \left(-\Delta \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} + \Delta \left(\frac{T'}{D}\right)^{\alpha/(\alpha+1)} \right) \\ &= \frac{\Delta}{Z} \cdot \left(\left(\frac{T'}{D}\right)^{\alpha/(\alpha+1)} - \left(\frac{T}{D}\right)^{\alpha/(\alpha+1)} \right) \end{aligned}$$

which again is strictly positive as long as $T' > T$ and $\Delta > 0$, the latter of which is satisfied by construction. \square

Overall, we see that in every non-trivial instantiation of the expressivity-aware quantization model, increasing expressivity improves $L(D)$, which closely approximates $\tilde{L}(D)$ for large enough D .

E.4 Main Results

Combining the results above on parameter (Corollary 5.1) and data scaling (Lemma 7), we obtain:

Theorem 4: Expressivity-Aware Scaling Laws

Consider the quantization model of neural scaling laws augmented by *Assumptions 1 to 3*. Then, as a function of the number of parameters N , the loss $\tilde{L}(N)$ is closely approximated by a power law $L(N)$, i.e., $\tilde{L}(N) \approx L(N)$ where:

$$L(N) - L_\infty^\epsilon \propto A_\epsilon \cdot N^{-\alpha}, \quad \text{where } A_\epsilon = (L_0 - L_\infty^\epsilon) \cdot (C + \epsilon(C' - C))^\alpha.$$

Similarly, as a function of the token budget D , the loss $\tilde{L}(D)$ is closely approximated by a power law $L(D)$, i.e., $\tilde{L}(D) \approx L(D)$ where

$$L(D) - L_\infty^\epsilon \propto B_\epsilon \cdot D^{-\alpha/(\alpha+1)}, \quad \text{where } B_\epsilon = (1 - \epsilon)\Delta T^{\alpha/(\alpha+1)} + \epsilon\Delta' T'^{\alpha/(\alpha+1)}.$$

Finally, the irreducible loss L_∞^ϵ for both power laws depends on expressivity via

$$L_\infty^\epsilon = L_0 - (1 - \epsilon)\Delta - \epsilon\Delta'.$$

Combining Corollary 5.2 and Corollary 7.1 yields:

Corollary 4.1: Expressivity Always Improves Scaling

Fix a nontrivial instantiation of *Assumptions 1 to 3*, i.e., where either $\Delta' < \Delta$, or $C' > C$ and $T' > T$. Then, both $L(N)$ and $L(D)$ strictly decrease as ϵ decreases, elementwise for all N, D .

The following follows straightforwardly from assuming $\Delta' < \Delta$:

Corollary 4.2: Expressivity Can Shift Irreducible Loss

If and only if $\Delta' < \Delta$, irreducible loss L_∞^ϵ strictly decreases as ϵ decreases.

Future Work. There are several extensions to this analysis worth pursuing. First, we have assumed learnable tasks are prioritized in order of frequency, whereas a more optimal policy would order them by *parameter efficiency* p_n/C_n . It would also be interesting to consider joint parameter-data scaling $L(N, D)$ rather than scaling laws derived in isolation. On the empirical side, one could test whether these scaling laws match real LMs in controlled settings where ϵ is known, or whether it is possible to isolate tasks in the data in practical settings. All of these directions would close the gap between the idealized quantization model and practical training, and could inform choices about the interaction between data and architecture during pretraining.

Table 21 Architecture ablation results – per-domain breakdown. Per-domain OLMOBASEVAL BPB for pure and hybrid architectures. **Bold** indicates best; underline second best; † best within section. ★ marks our selected architecture. Note that per-size comparisons should be interpreted with care, as architectures at the same nominal scale differ in actual parameter count (see Table 22).

Small scales (60M–370M parameters):

Architecture	Attn %	60M			100M			190M			370M		
		Math	Code	QA	Math	Code	QA	Math	Code	QA	Math	Code	QA
<i>Pure Architectures</i>													
Transformer	100%	1.102	0.931	1.433	0.981	0.804	1.326	0.887	0.707	1.257	0.782	0.616	1.119
GDN	0%	1.040 [†]	<u>0.837</u>	1.363 [†]	0.949 [†]	0.752 [†]	1.270 [†]	0.839 [†]	0.671 [†]	1.176 [†]	0.746 [†]	0.573	1.066
Mamba2	0%	1.110	0.917	1.411	0.994	0.798	1.316	0.896	0.711	1.215	0.790	0.616	1.122
<i>Hybrid: Interleaved Attention</i>													
GDN (1:1)	50%	1.044	0.861	1.374	0.936	0.765	1.273	0.839	0.673	1.178	0.744	0.585	1.083
GDN (3:1)★	25%	1.034	0.849 [†]	1.349	0.934	0.752 [†]	1.271	0.835	0.663	1.177	0.741 [†]	0.588	1.069 [†]
GDN (7:1)	12.5%	1.032 [†]	0.850	1.365	0.932 [†]	0.764	1.269 [†]	0.840	0.668	<u>1.169</u>	0.745	0.584 [†]	1.080
Mamba2 (3:1)	25%	1.080	0.905	1.404	0.957	0.786	1.292	0.863	0.698	1.202	0.770	0.603	1.110
<i>Hybrid GDN (3:1): Middle Placement</i>													
Interleaved★	25%	1.034 [†]	0.849 [†]	<u>1.349</u>	0.934 [†]	0.752 [†]	1.271 [†]	<u>0.835</u>	<u>0.663</u>	1.177 [†]	0.741 [†]	0.588	1.069 [†]
Middle	25%	1.041	0.868	1.353	0.934	0.754	1.275	0.842	0.671	1.183	0.743	0.576 [†]	1.081
<i>Hybrid GDN (3:1): Gate/Eigenvalue Ablations</i>													
Neg EV, gate★	25%	1.034	0.849	1.349 [†]	0.934	0.752	1.271	0.835 [†]	0.663 [†]	1.177	0.741	0.588	1.069
Neg EV, no gate	25%	<u>1.026</u>	0.838 [†]	1.378	0.930	0.733	1.264	0.837	0.673	1.171 [†]	<u>0.739</u>	0.595	1.070
Pos EV, gate	25%	1.033	0.849	1.376	0.928	0.746	<u>1.267</u>	0.839	0.675	1.176	0.764	0.598	1.122
Pos EV, no gate	25%	1.037	0.843	1.366	<u>0.929</u>	0.757	1.278	0.839	0.681	1.183	0.738	0.579 [†]	1.067 [†]
<i>Pure GDN: Gate/Eigenvalue Ablations</i>													
Neg EV, gate	0%	1.040	0.837	1.363	0.949	0.752	1.270 [†]	0.839 [†]	0.671	1.176	0.746	<u>0.573</u>	<u>1.066</u>
Pos EV, gate	0%	1.026	0.834	1.352 [†]	0.936 [†]	<u>0.740</u>	1.271	0.848	0.664 [†]	1.167	0.745 [†]	0.577	1.073
Neg EV, no gate	0%	1.056	0.846	1.360	0.958	0.762	1.276	0.857	0.678	1.176	0.757	0.599	1.071
Pos EV, no gate	0%	1.052	0.856	1.369	0.963	0.751	1.292	0.851	0.683	1.170	0.758	0.585	1.075

Large scales (600M–1B parameters):

Architecture	Attn %	600M			760M			1B		
		Math	Code	QA	Math	Code	QA	Math	Code	QA
<i>Pure Architectures</i>										
Transformer	100%	0.724	0.564	1.055	0.686	0.538	1.015	0.622 [†]	0.492	0.933
GDN	0%	0.705 [†]	0.551 [†]	1.026 [†]	0.673 [†]	0.522 [†]	0.970 [†]	0.624	0.485 [†]	0.921 [†]
Mamba2	0%	0.736	0.577	1.047	0.699	0.547	1.005	0.662	0.524	0.969
<i>Hybrid: Interleaved Attention</i>										
GDN (1:1)	50%	0.725	0.571	1.065	0.663	0.527	0.983	0.610	0.484 [†]	0.928
GDN (3:1)★	25%	0.694	0.546	1.022	0.658 [†]	0.523 [†]	0.969	0.604	0.488	0.914 [†]
GDN (7:1)	12.5%	0.693	<u>0.543</u>	<u>1.009</u>	0.660	0.524	0.978	0.612	0.486	0.927
Mamba2 (3:1)	25%	0.714	0.561	1.042	0.673	0.531	0.992	0.638	0.509	0.946
<i>Hybrid GDN (3:1): Middle Placement</i>										
Interleaved★	25%	0.694 [†]	0.546 [†]	1.022	0.658 [†]	0.523	<u>0.969</u>	<u>0.604</u>	0.488	0.914 [†]
Middle	25%	0.698	0.547	1.015 [†]	0.661	0.521 [†]	0.982	0.608	0.487 [†]	0.922
<i>Hybrid GDN (3:1): Gate/Eigenvalue Ablations</i>										
Neg EV, gate★	25%	0.694	0.546	1.022	0.658	0.523	0.969 [†]	0.604 [†]	0.488	0.914
Neg EV, no gate	25%	0.696	0.548	1.012 [†]	0.654	0.526	0.983	0.610	0.483	0.904
Pos EV, gate	25%	0.695	0.546	1.030	0.659	0.514	0.970	0.605	<u>0.484</u>	<u>0.912</u>
Pos EV, no gate	25%	<u>0.694</u>	0.545 [†]	1.021	<u>0.656</u>	0.518	0.977	0.605	0.486	0.918
<i>Pure GDN: Gate/Eigenvalue Ablations</i>										
Neg EV, gate	0%	0.705	0.551	1.026	0.673	0.522	0.970 [†]	0.624	0.485 [†]	0.921
Pos EV, gate	0%	0.705 [†]	0.542	1.004 ₆₉	0.669 [†]	<u>0.516</u>	0.975	0.614 [†]	0.492	0.917 [†]
Neg EV, no gate	0%	0.710	0.546	1.026	0.679	0.527	0.986	0.623	0.489	0.929
Pos EV, no gate	0%	0.712	0.559	1.024	0.677	0.526	0.977	0.626	0.488	0.919

Table 22 Architecture configurations and non-embedding parameter counts (millions). d : model dimension, h : number of attention heads, l : number of layers. Architectures at the same nominal scale can differ substantially in parameter count due to differences in layer composition.

Architecture	Attn %	60M	100M	190M	370M	600M	760M	1B
d		384	512	768	1024	1280	1536	2048
h		8	8	12	16	16	16	16
l		8	12	12	16	16	16	16
<i>Pure Architectures</i>								
Transformer	100%	57	102	190	371	548	758	1279
GDN	0%	78	140	276	574	780	1011	1549
Mamba2	0%	61	110	207	410	606	841	1423
<i>Hybrid: Interleaved Attention</i>								
GDN (1:1)	50%	68	121	233	472	664	885	1414
GDN (3:1) [★]	25%	73	130	254	523	722	948	1482
GDN (7:1)	12.5%	75	133	262	548	751	980	1516
Mamba2 (3:1)	25%	60	108	203	400	592	820	1387
<i>Hybrid GDN (3:1): Middle Placement</i>								
Interleaved (reference) [★]	25%	73	130	254	523	722	948	1482
Middle	25%	70	127	247	510	707	932	1465
<i>Hybrid GDN (3:1): Gate/Eigenvalue Ablations</i>								
Neg EV, gate [★]	25%	73	130	254	523	722	948	1482
Neg EV, no gate	25%	73	130	254	523	722	948	1482
Pos EV, gate	25%	73	130	254	523	722	948	1482
Pos EV, no gate	25%	73	130	254	523	722	948	1482
<i>Pure GDN: Gate/Eigenvalue Ablations</i>								
Neg EV, gate	0%	78	140	276	574	780	1011	1549
Pos EV, gate	0%	78	140	276	574	780	1011	1549
Neg EV, no gate	0%	78	140	276	574	780	1011	1549
Pos EV, no gate	0%	78	140	276	574	780	1011	1549