A FINE-GRAINED HIERARCHY AND BENCHMARK FOR STATE TRACKING IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Theoretical limitations of transformers on *state tracking*—the ability to track information about entities or variables over long contexts—have motivated new recurrent language modeling (LM) architectures such as Gated DeltaNet (GDN). These architectures have, in particular, been benchmarked against the S_5 word problem, a theoretically hard state-tracking task that captures abilities such as code evaluation under arbitrary variable assignment. It, however, remains open whether other, simpler forms of state tracking might both be more relevant in practice and already explain observed differences between these architectures. We propose evaluating state tracking using a hierarchy of word problems with four increasing levels of difficulty: *r-trivial*, corresponding to immutable assignment to variables in programming; *aperiodic*, where assignments are mutable; *soluble*, where assignments are self-referential; and *general* finite state tracking. Theoretically, we show that the class of problems transformers are thought to length-generalize on does not even contain all aperiodic state tracking. On the other hand, we prove that GDN with positive eigenvalues expresses precisely the class of all aperiodic problems, and the regular languages recognized by certain transformers are precisely all soluble ones, thus providing a fine-grained picture of state tracking capabilities of various architectures. Taking these ideas to practice, we construct **StateBench**, a 4-tier long-horizon state-tracking benchmark. Our systematic direct training experiments across architectures largely align with theoretical expectation, albeit with interesting observations at very large scale length-generalization. Additionally, unlike prior work, we also evaluate pretrained models—both open hybrid and closed frontier ones—on StateBench, finding that none of them robustly solves even soluble state tracking, suggesting a direction for future improvement.

1 INTRODUCTION

Tracking the state of the world across a long context is necessary for understanding discourse, documents, and code (Kamp, 1981; Kim & Schuster, 2023; Hsieh et al., 2024). Prior work has examined the difficulty of state-tracking tasks by viewing them as equivalent to word problems for finite monoids, and relating the algebraic properties of these structures to their computability by various neural architectures (Liu et al., 2023; Merrill et al., 2024). While much focus has been given to the hardest class of state-tracking problems, such as the S_5 word problem, less attention has been given to intermediate levels of problem difficulty, which may prove more representative of the kinds of real-world state-tracking tasks language models are expected to solve and might already explain observed performance differences across architectures.

To better understand finegrained state-tracking differences between architectures, we propose evaluating state tracking capabilities across four hierarchical levels naturally motivated by the classical Krohn–Rhodes decomposition for finite monoids (Krohn & Rhodes, 1965): *general finite*, involving arbitrary transpositions of variables; *soluble*, involving variable assignment by self-reference; *aperiodic*, involving the reassignment of variables to new values; and *r-trivial*, involving constant registers which can only be written to once. We then evaluate the theoretical expressivity and empirical performance of various language model architectures across levels of this hierarchy.

We first theoretically explore the ability of Transformers (Vaswani et al., 2017) and Gated DeltaNet (GDN; Yang et al. 2024) models to represent state tracking problems across this hierarchy and the

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

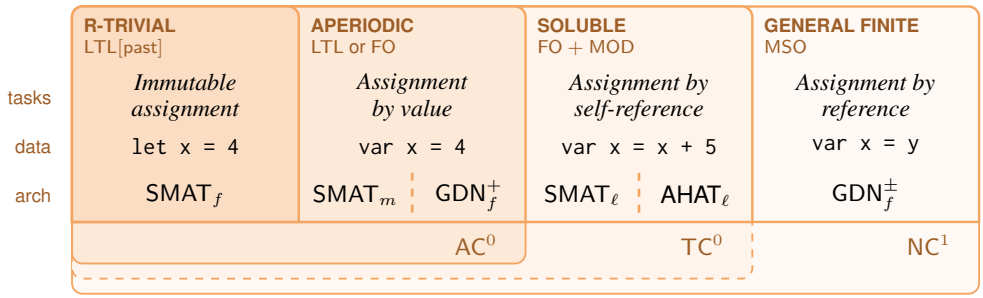

Figure 1: Placement of language model architectures within the hierarchy of state-tracking (monoid word problem) complexity, along with examples of the kinds of tasks that fall into these categories. SMAT_m and SMAT_ℓ each lie partially inside the respective monoid class but are not equal to it. Areas below the horizontal line indicate circuit complexity classes that, when restricted to regular languages, are equal to the monoid classes. The dashed line represents the conjecture $\mathsf{TC}^0 \neq \mathsf{NC}^1$.

role of particular hyperparameters. Complementing existing results relating low-precision softmax transformers to r-trivial and aperiodic state tracking (Li & Cotterell, 2026; Yang et al., 2025), we show that the state tracking problems expressible by log-precision averaging hard-attention transformers, a well-studied theoretical model, are exactly the soluble ones. Next, whereas GDN with an extended eigenvalue spectrum can represent all state tracking tasks, we characterize the state tracking capabilities of GDN with positive eigenvalues as *exactly* aperiodic monoids. Synthesizing prior work and our new results reveals finegrained distinctions between the theoretical state tracking capabilities of transformers and GDN, as well as the role of precision and parameterization details on the state tracking expectations for each architecture.

Motivated by these theoretical results, we introduce StateBench, a long-context evaluation benchmark with state-tracking problems at varying levels of complexity and supporting evaluation of very large scale length-generalization (up to 64K state changes). We evaluate models trained directly on StateBench data, showing that models containing recurrent modules length-generalize fully on r-trivial aperiodic data and show limited generalization on general finite data, while transformers fail to generalize on r-trivial data and fail to learn even in-distribution general finite data. On open-source pretrained models including two Olmo 3 7B variants, we find that all architectures do well on r-trivial data, degrade meaningfully on aperiodic data, and fail completely on general finite data. We find that a variety of frontier proprietary Gemini, Claude, and GPT LLMs can solve r-trivial and aperiodic tasks well but exhibit strong degradation on soluble and general finite tasks as a function of sequence length, dropping to near-zero performance on the hardest problems in StateBench. Lastly, we contextualize these results by connecting monoid periodicity to the sensitivity of boolean functions, showing that proper understanding of many tasks involving real-world source code requires state-tracking capabilities beyond the expressivity and learned capacity of current language models.

2 A Hierarchy of Finite State Tracking

To analyze the difficulty of state-tracking problems and connect their solvability to neural architectures, we construct a formal model of them based on an algebraic object known as a *(finite) monoid*, defined as a triple $(M, \circ, 1)$ of some (finite) set M, a composition operation $\circ$, and a neutral identity element 1. Prior work has noted that monoids are a natural model of state tracking by means of their composition operator (Liu et al., 2023; Merrill et al., 2024; Nowak et al., 2026a;b, *inter alia*). A *word* w over M is a string $w_1 w_2 \dots w_n$ of elements of M concatenated together, representing the composition $w_1 \circ w_2 \circ \dots \circ w_n$. Intuitively, w represents a sequence of updates to a system: given some initial state s_0, observing w_1 moves the system to state s_1; observing w_2 moves it to s_2; and so forth. Since M is closed under composition, the word $w = w_1 w_2 \dots w_n$ is itself an element of M; computing exactly which element is known as the *word problem*[1] on M.

[1]For brevity, we will often omit writing the operator $\circ$ explicitly, treat composition as concatenation, and refer to a monoid $(M, \circ, 1)$ simply by its set M. As a canonical example, consider the integers $(\mathbb{Z}, +, 0)$ with addition; the word problem on $\mathbb{Z}$ is identifying $a_1 + a_2 + \dots + a_k$ for any $a_1, \dots, a_k \in \mathbb{Z}$.

We will formalize different levels of complexity for state tracking in terms of word problems for different types of monoids. A useful source of intuition for this will be the different kinds of variable state manipulation that are possible in programming. We can distinguish between four kinds of state updates: (i) *immutable writes*, such as defining a constant value like `let x = 4`; (ii) *mutable writes*, such a (re)defining a variable value like `var x = 4; ... x = 2`; (iii) *self-referential writes*, such as incrementing a variable like `var x = 4; x += 3`; and (iv) *referential writes*, such as swapping two variables like `var x = 4; var y = 2; (x,y) = (y,x)`.

By modeling state-tracking as a word problem on a finite monoid, we can relate these various kinds of state update operations to the algebraic structure of the monoid and use this to understand the computational difficulty of tracking the state-space—calculating the value of the system after observing an arbitrary sequence of updates. The well-known eponymous theorem of Krohn & Rhodes (1965) establishes that every finite monoid M can be decomposed into the iterated wreath product, denoted $M_1 \wr M_2 \wr \cdots \wr M_k$, of finitely-many irreducible factors, loosely analogous to a (non-unique) prime factorization.[2] Each irreducible factor M_i in this decomposition is either the two-element semilattice $U_1 \triangleq \{1, 0\}$ where $0x = 0$, corresponding to *immutable writes*; the "flip-flop" monoid $U_2 \triangleq \{1, a, b\}$ where $xy = y$ for $x, y \in \{a, b\}$, corresponding to *mutable writes* by value; the commutative group $\mathbb{Z}_p$ of integers modulo p for some prime p, corresponding to *self-referential writes*; or a non-commutative finite simple group, corresponding to *referential writes*.

Different kinds of state tracking can be formalized based on the kinds of finite monoids they represent. A finite monoid M is called (i) *r-trivial* if for all $x, y \in M$, $xM = yM$ implies $x = y$ (Green, 1951); (ii) *aperiodic* if for every $x \in M$ there is some $n \geq 1$ with $x^n = x^{n+1}$ (Schützenberger, 1965); and (iii) *soluble* if every subgroup of M is soluble (Barrington & Thérien, 1988). For our purposes, however, a more useful and equivalent characterization of these types of monoids is terms of the kinds of core operations that appear in their Krohn–Rhodes (KR) decomposition:

Proposition 1 (Monoid types, informal; cf. Appendix A for a formal statement)**.** *A finite monoid M with KR decomposition $M_1 \wr M_2 \wr \cdots \wr M_k$ is* (i) r-trivial *if every M_i is U_1;* (ii) aperiodic *if every M_i is U_2;*[3] *and* (iii) soluble *if every M_i is either U_2 or $\mathbb{Z}_p$ for some prime p.*

As illustrated in Figure 1, these classes form a nested hierarchy: $\mathsf{Rtrv} \subsetneq \mathsf{Aprd} \subsetneq \mathsf{Sol} \subsetneq \mathsf{Fin}$, where Rtrv is the class of word problem languages[4] whose monoid is r-trivial, Aprd is the class for aperiodic monoids, Sol is the class for soluble, and Fin contains all word problem languages on finite monoids including S_5. Thus, Rtrv can be understood as a generalized class of all state tracking involving only immutable writes, Aprd as a natural class of state tracking problems involving only writes "by value", Sol as a harder class of state tracking where interaction between variables is limited, and Fin as the class of unconstrained finite state tracking problems.

Correspondence to Formal Language, Logic, and Circuit Complexity Classes. In addition to having intuitive interpretations in terms of variable assignments, these levels within the state tracking hierarchy coincide with other notable classes of problems. From a formal language theory perspective, Fin is exactly the class of regular languages, and Aprd is exactly the class of star-free languages (Schützenberger, 1965). From a logical perspective, $\mathsf{Fin} = \mathsf{MSO}$, the languages definable in monadic second-order logic (Büchi, 1990), whereas $\mathsf{Aprd} = \mathsf{FO}[<] = \mathsf{LTL}$: it corresponds to the languages definable in first-order logic (McNaughton & Papert, 1971) or, equivalently, linear temporal logic (Kamp, 1968). $\mathsf{Sol} = \mathsf{FO}[<]+\mathsf{MOD}$, the languages definable in first-order logic augmented with modular predicates (Tesson & Therien, 2007), and $\mathsf{Rtrv} = \mathsf{LTL}[\mathsf{past}]$, the languages definable in the restricted past fragment of $\mathsf{FO}[<]$ or LTL (Li & Cotterell, 2026; Jerad et al., 2025).

Finally, from a circuit complexity perspective, Aprd corresponds to the regular languages in AC^0 (Mix Barrington et al., 1992, Theorem 3), Fin to the regular languages in NC^1 (all of them), and Sol to the regular languages in ACC^0 as well as those in TC^0, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$ (Mix Barrington et al., 1992, Theorem 8). The convergence of these classes with other areas suggests the hierarchy reflects fundamental distinctions in state tracking capabilities relevant across computational models.

[2] We defer formal definitions of the algebraic concepts related to monoids and Krohn–Rhodes decomposition to Appendix A, focusing here on developing the intuition behind the state tracking hierarchy.

[3] U_1 is a submonoid of U_2, obtained by setting $a = b = 0$. U_2 thus includes U_1 in a natural way.

[4] For a monoid M, w is in the word problem language $L(M)$ if the product of w evaluates to 1.

3 EXPRESSIVITY OF ARCHITECTURES FOR STATE TRACKING

We now explore the state tracking abilities of various transformer and recurrent architectures, in relation to our state tracking hierarchy. Our goal is to characterize models based on whether or not they can express a length-general solution to the word problem for the particular class.

Transformers and GDN. We adopt the formal model of Merrill et al. (2026a) and Merrill (2025); see § A.2 for formal definitions. Each architecture uses a stack of d layers, interleaving tokenwise feedforward sublayers with sequence-mixing sublayers that are defined as follows:

- Softmax attention **(SMAT)**: the standard causally-masked attention of Vaswani et al. (2017), where token i outputs $\sum_{j \leq i} \mathrm{softmax}_j(\mathbf{q}_i^\top \mathbf{k}_j)\, \mathbf{v}_j$.
- Averaging hard attention **(AHAT)**: an idealization in which token i attends uniformly to tied positions $j \leq i$ that maximize $\mathbf{q}_i^\top \mathbf{k}_j$ and averages their values (Pérez et al., 2021).
- Gated DeltaNet **(GDN)** (Yang et al., 2024): a linear recurrence over a matrix-valued state,

$$\mathbf{S}_t = \alpha_t\big(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top\big)\mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top, \qquad \mathbf{o}_t = \mathbf{S}_t^\top \mathbf{q}_t,$$

  with gate $\alpha_t \in [0,1]$ and write strength β_t. The eigenvalues of the transition $\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ are 1 and $1-\beta_t$. Following Grazzi et al. (2025), we write GDN^+ when $\beta_t \in [0,1]$ (nonnegative eigenvalues) and $\mathsf{GDN}^\pm$ when $\beta_t \in [0,2]$ (eigenvalues in $[-1,1]$).

The expressive power of these models depends on the numerical precision of the network's activations. Following Merrill et al. (2026a), we distinguish *fixed* precision (f; a fixed number of bits w.r.t. input length n), *log* precision (ℓ; $O(\log n)$ bits, enough to represent positions and uniform averages over the context), and *mixed* precision (m; fixed precision everywhere except inside attention aggregation, as in Yang et al. 2025). We write, e.g., SMAT_f for the class of languages recognized by fixed-precision SMATs. Throughout, we say an architecture *solves* the word problem over M if a single model in that architecture computes the correct output for every word over M.

3.1 KNOWN EXPRESSIVITY RESULTS

Transformers. We first review some known results relating softmax transformers (SMAT) to different levels in the state tracking hierarchy, parameterized by different levels of precision (finite f, mixed m, or log ℓ). First, Li & Cotterell (2026) and Jerad et al. (2025) showed $\mathsf{SMAT}_f = \mathsf{Rtrv}$, i.e., fixed-precision SMAT solves precisely the word problem over r-trivial monoids.[5] Second, Yang et al. (2026) showed that the language C-RASP, often used as an abstraction of transformers, is equivalent to the wreath product closure of bounded-depth Dyck languages, which is a strict subset of aperiodic. C-RASP is known to be equivalent to mixed-precision SMATs (Yang et al., 2025). It follows that $\mathsf{SMAT}_m \subset \mathsf{Aprd}$, i.e., only for a strict subset of aperiodic monoids M can mixed-precision transformers solve the word problem over M. Thus, while going from finite precision to mixed precision increases SMAT expressivity above Rtrv, mixed-precision SMATs remain properly within Aprd. Third, even in the log-precision regime, both SMAT and AHAT are known to be in the circuit class uniform TC^0 (Merrill & Sabharwal, 2023a). Moreover, under the circuit complexity assumption $\mathsf{TC}^0 \neq \mathsf{NC}^1$, the regular languages in TC^0 are exactly Sol (Mix Barrington et al., 1992, Theorem 8). Thus, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, we have both $\mathsf{SMAT}_\ell \cap \mathsf{Fin} \subseteq \mathsf{Sol}$ and $\mathsf{AHAT}_\ell \cap \mathsf{Fin} \subseteq \mathsf{Sol}$. Overall, while more precision may increase the state tracking capabilities of SMAT, they remain within Sol, and it has thus far been unknown whether they fully realize Sol.

GDN. Beyond transformers, we consider some partial results about GDNs. Grazzi et al. (2025) derived an upper bound of (word problems over) aperiodic monoids for fixed-precision GDN^+, i.e., $\mathsf{GDN}^+_f \subseteq \mathsf{Aprd}$. Merrill et al. (2026a) show $\mathsf{GDN}^\pm_f = \mathsf{Fin}$, i.e., for any finite monoid, there exists a fixed-precision GDN (with both positive and negative eigenvalues) that solves the word problem over it.[6] It's worth noting that this captures the full gamut of finite state tracking problems, implying, in particular, that hybridization of fixed-precision $\mathsf{GDN}^\pm$ with SMAT or AHAT does not further increase the architecture's expressivity for state tracking.

[5] Technically, this characterization is sensitive to the arithmetic model: Nowak et al. (2026b) show that under standard left-to-right floating-point summation, fixed-precision softmax attention can exceed Rtrv and recognize some aperiodic languages, while variants of attention give definite, r-trivial, or locally r-trivial classes. Across these arithmetic models, then, $\mathsf{Rtrv} \subseteq \mathsf{SMAT}_f \subseteq \mathsf{Aprd}$.

[6] This result relies on pre-pending a fixed, input-independent prefix to the input.

3.2 New results leveraging Krohn–Rhodes decomposition

We now leverage the Krohn–Rhodes view of state tracking to derive several new results placing transformer and GDN variants at various levels of the state-tracking hierarchy. These results help relate the state tracking capabilities of the two different architectures and elucidate the role of key parameters like precision and attention type for state tracking. Proofs are deferred to Appendix B throughout.

The following lemma provides the foundation for our formal results, identifying the "building block" monoids (U_1, U_2, and $\mathbb{Z}_p$) that various architectures can implement. Here, "implement" means that given a one-hot representation of a monoid element w_i in the input-layer hidden state at token i, there will be a subspace of the representation of token i in the final layer that encodes the partial product $w_1 w_2 \dots w_i$ as a one-hot vector.

Lemma 1 (Implementing Basic Monoids)**.** *Various attention and recurrent mechanisms can solve various basic finite monoid word problems, as follows:*

a. There exists a one layer SMAT$_f$ that implements U_1.
b. There exists a one layer GDN$_f^+$ that implements U_2.
c. There exists a one layer AHAT$_\ell$ that implements U_2.
d. For any prime p, there exists an 8-layer AHAT$_\ell$ that implements $\mathbb{Z}_p$.

Lemma 1 shows that for each basic monoid, there is a block of layers of various types of architectures that can be used as a gadget to implement that monoid. Moreover, the iterated wreath product of k basic monoids can be implemented by a depth-wise composition of the corresponding gadgets:

Lemma 2 (Implementing Monoid Composition)**.** *Let M be a finite monoid with a KR decomposition $M_1 \wr M_2 \wr \cdots \wr M_k$. Suppose that for each i, an attention-based or recurrent architecture A_i of depth d_i can implement M_i. Let $d \triangleq \sum_{i=1}^{k} d_i$. Then the depth-$d$ composition $A_k \circ \ldots \circ A_2 \circ A_1$ of the architectures implements M.*

Armed with these lemmas, we are ready to state our main technical findings, in which M refers to a finite monoid. Recall (§ 3.1) that Grazzi et al. (2025) showed GDN$_f^+$ can only express aperiodic monoid word problems. Our first result strengthens this to an exact characterization:

Theorem 1 ($\mathsf{GDN}_f^+ = \mathsf{Aprd}$)**.** *There exists a fixed-precision GDN$^+$ that solves the word problem over M if and only if M is aperiodic.*

The idea is that aperiodic monoids can be decomposed into flip-flops, implying via Lemma 1.b that GDN$_f^+$ can solve all of Aprd. For any aperiodic monoid, our lower bound constructs a GDN$^+$ that has the same depth as the number of basic monoids in a KR decomposition of the monoid. Our result holds for (ungated) DeltaNet in addition to GDN, but we state it with the more common GDN acronym. Thus, gating does not actually change the expressive power of fixed-precision DeltaNet, though it may affect the succintness of representation for various problems or learning dynamics.

Next, we leverage Krohn–Rhodes in a similar way to obtain a lower bound on log-precision AHAT:[7]

Theorem 2 ($\mathsf{AHAT}_\ell \supseteq \mathsf{Sol}$)**.** *If M is soluble, then there exists a log-precision AHAT that solves the word problem over M.*

As discussed in § 3.1, assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, $\mathsf{AHAT}_\ell \cap \mathsf{Fin} \subseteq \mathsf{Sol}$. Since $\mathsf{Sol} \subseteq \mathsf{Fin}$, Theorem 2 implies that, under this circuit complexity assumption, $\mathsf{AHAT}_\ell \cap \mathsf{Fin}$ is precisely Sol, i.e., the regular languages recognized by log-precision AHAT are exactly those whose underlying monoid is soluble:

Corollary 1 ($\mathsf{AHAT}_\ell \cap \mathsf{Fin} \approx \mathsf{Sol}$)**.** *Assuming $\mathsf{TC}^0 \neq \mathsf{NC}^1$, $\mathsf{AHAT}_\ell \cap \mathsf{Fin} = \mathsf{Sol}$, i.e., there exists a log-precision AHAT that solves the word problem for M if and only if M is soluble.*

A picture of architectures' relative state tracking capabilities emerges from these results. For GDN, these capabilities are controlled by the eigenvalues, not gating: GDN$^+$ is limited to aperiodic problems while GDN$^\pm$ covers all finite monoids. On the other hand, AHAT with log precision sits in-between, solving word problems over soluble monoids. For SMAT, precision controls expressivity, with mixed- and fixed-precision transformers corresponding to proper subclasses of aperiodic. The exact expressivity of log-precision SMAT or GDN$^+$ on state tracking problems remains open.

[7] A similar result in a nonuniform transformer model was given by Liu et al. (2023, Theorem 2), though it was nontrivial to extend their result to the uniform AHAT model.

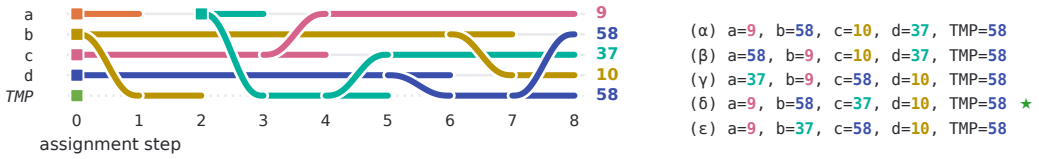

Figure 2: **left:** A visualization of an 8-assignment general finite sequence with $4+1$ variables. Each variable is assigned an initial value (e.g., $\mathsf{b} \leftarrow 10$, $\mathsf{c} \leftarrow 9$). At step t, a variable may be overwritten by a new constant value (as in $\mathsf{a} \leftarrow 37$ at t_2) or it may be swapped with another (as b and d are swapped between t_6 and t_8). **right:** Five possible final states for the sequence, with the correct one (δ) annotated with a green star.

Another interesting open question is: *Could a GDN$^+$ based hybrid model, possibly under higher precision, solve word problems over soluble monoids?* Krohn–Rhodes theory lets us derive a conditional result. For any prime p, suppose we have an architecture GM^p (a *mod gadget*) that, given a number $z \in \mathbb{Z}_{\geq 0}$ (represented as $1/z$ or layer-norm hash $\phi(z)$; Merrill & Sabharwal 2023b) as input, can compute a representation of $z \bmod p$. Then we can leverage KR decompositions to show that a hybrid model consisting of this gadget, along with GDN$^+$ and SMAT, covers *all* soluble monoids:

Proposition 2. *Hybrid* $(\mathsf{GDN}^+_f : \mathsf{SMAT}_\ell : \mathsf{GM}) \supseteq \mathsf{Sol}$*, i.e., for any soluble M, there exists a hybrid model composed of fixed-precision GDN$^+$, log-precision SMAT, and mod gadgets GM^p for a fixed set of p that solves the word problem over M.*

It remains open whether GDN$^+$ with mixed- or log-precision is contained in aperiodic or, on the other hand, can implement the mod operation; if it's the latter, Proposition 2 shows that hybrid GDN models would capture all of soluble.

4 The StateBench Dataset

To evaluate how well language models can learn and solve state-tracking problems we introduce StateBench, a new benchmark of synthetic state-tracking data that can be used to generate training and evaluation samples at targeted levels of algebraic complexity. Each StateBench problem is a monoid word problem over m variables, initialized to starting values and updated with n assignments. Assignments may be transpositions which swap two variables (finite), self-increment (soluble), reassignments by value (aperiodic), or constant assignments (r-trivial; see Figure 1). Each sequence is paired with one correct and four incorrect candidate final states (Figure 2), supporting use as both a generative and a multiple-choice evaluation.

We first generate a general finite state tracking sample that mixes assignments-by-reference ($x \leftrightarrow y$, implemented with a temporary variable) with aperiodic assignments-by-value to a new constant value. For each sample, the number of aperiodic assignments is drawn log-uniformly from $[1, 64]$. From this general finite state tracking source task, we derive a soluble sample by replacing all assignments-by-reference with the equivalent self-increment value ($x \leftarrow x + \delta \bmod N$, for some max value N), an aperiodic sample by replacing it with the value it would have copied ($x \leftarrow K$), and an r-trivial sample by replacing every assignment with the variable's initial value ($x \leftarrow x_0$). Deriving all four from one source controls for the number of variables, the number of distinct values, and the positions of assignments across complexity levels.

We create a static evaluation set containing r-trivial, aperiodic, soluble, and finite samples with 8 to 64,000 assignments. Each sample is annotated with its complexity class, m, n, sensitivity, and length in Olmo-3 tokens; see Table 1 in Appendix D. Since pretrained models may be sensitive to surface form, each sample is rendered in six textual formats that share the same underlying algebraic structure (Appendix C).

5 Directly-Trained Model Evaluations

We first validate our theoretical placement of various language model architectures on the state-tracking hierarchy by training models directly on data from StateBench and evaluating their ability

to generalize to longer sequences. We consider six architectures: Transformers (Vaswani et al., 2017) with rotary position embedding (RoPE; Su et al. 2021) and without any positional embedding (NoPE; Wang et al. 2024); Gated DeltaNet (GDN; Yang et al. 2024) with only positive eigenvalues ($+$) and with both positive and negative eigenvalues ($\pm$); and Transformer-NoPE–GDN hybrid models with only positive eigenvalues ($+$) and both positive and negative eigenvalues ($\pm$) eigenvalues (Merrill et al., 2026b). For each architecture we randomly initialize 275M-parameter models and train them with a language modeling objective on sequences of a StateBench prefix and its valid completion. We train each model on $1\times$ the compute-optimal amount of data for pretraining following the "Chinchilla" methodology of Hoffmann et al. (2022). We consider two aspects of performance: first, whether models are able to *learn* the task sufficiently well to attain near-ceiling performance on in-distribution numbers n of assignments; and second, whether the models are able to *generalize* to longer sequences with larger n. We train models with n between 8 and 256, and evaluate them on sequences with n up to 64,000 (i.e., $250\times$ longer), looking in particular at generalization up to $4\times$ the training length as well as all the way to 64,000.

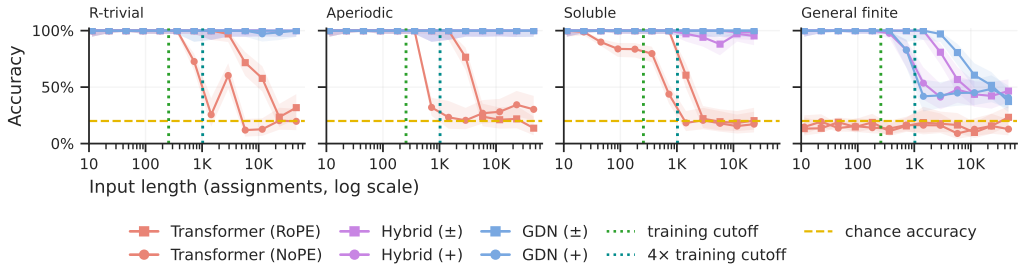

Figure 3: Direct Training: GDNs generalize perfectly on r-trivial, aperiodic, and soluble data, and partially on general finite data; Transformers fail to generalize beyond their training distribution on r-trivial, aperiodic, and soluble data and fail to even learn in-distribution general finite data. Error bars show Wilson confidence intervals.

Figure 3 shows that language model architectures show clear differences in their ability to learn state tracking at in-distribution lengths as well as to generalize to longer sequences. **Transformers** learn in-distribution *r-trivial* and *aperiodic* data well but do not generalize perfectly beyond the training distribution; NoPE models degrade immediately beyond the training cutoff, while RoPE models generalize to roughly $10\times$ the training length before degrading. On *general finite* data, transformers do not even completely learn the in-distribution data. **GDN** models generalize perfectly on *r-trivial*, *aperiodic*, and *soluble* data. They also perfectly learn the training distribution for *general finite* data but fail to generalize perfectly beyond the training cutoff: GDN$^{\pm}$ models generalize to roughly $10\times$ the training lengths, whereas GDN^{+} models degrade immediately beyond the training cutoff. **Hybrid** models pattern like their GDN components, generalizing near-perfectly on r-trivial, aperiodic, and soluble data; Hybrid$^{\pm}$ models generalize longer than Hybrid^{+} ones on general finite data.

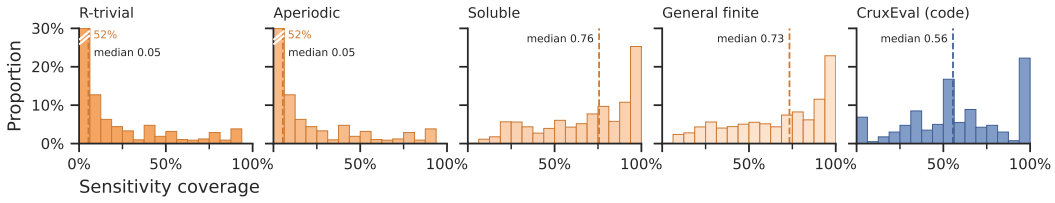

Figure 4: Sensitivity Analysis: Histograms of sample sensitivities in **(left)** the four complexity distributions in StateBench and **(right)** code samples from CruxEval.

Sensitivity makes tracking state harder. To interpret models' lackluster performance on general finite problems, we draw a connection between the periodicity of samples and their *sensitivity*, a well-studied complexity measure that describes how much perturbations in a function's input affect its output. Theoretically, monoid word problems have $o(n)$ sensitivity if they are aperiodic and $\Omega(n)$ otherwise (Amiri et al., 2025, Remark C.2). In other words, a state tracking problem is highly sensitive if and only if the underlying monoid is periodic.

We first measure the sensitivity of our derived evaluation samples and compare it to that of samples from real-world code evaluation data from CruxEval (Gu et al., 2024), a code-understanding benchmark of Python methods paired with canonical inputs and their expected outputs. In both cases, we measure sensitivity by calculating at how many steps during a program's execution modifying a variable's value could affect the final output.[8] As Figure 4 shows, r-trivial and aperiodic data has low average sensitivity, while soluble and general finite sequences are far more sensitive, validating that they truly reflect periodic state tracking. Code samples from CruxEval show a split distribution, with peaks at very high and very low sensitivities and a smoother distribution of sensitivities in the middle. Due to the correspondence between sensitivity and periodicity, this suggests periodic state tracking is important for evaluating natural code.

We next compare performance on general finite problems with per-sample sensitivity, finding a clear negative relationship: even among models that can learn general finite data in-distribution, samples with higher sensitivity are harder (cf. Figure 7 in Appendix D). This is aligned with prior observations that highly-sensitive functions are difficult for Transformers to learn (Hahn & Rofin, 2024). A data curriculum may aid learning sensitive state tracking problems (Siems et al., 2026; Barak et al., 2022; Kearns, 1998). To test if this helps for our models, we compare the results of training on randomly-shuffled general finite sequences to the same training distribution ordered by ascending sensitivity. We find, however, that this has negligible effects on GDN and Hybrid models and on Transformers with NoPE (cf. Figure 8 in Appendix D), and only a weak positive effect on Transformers with RoPE on in-distribution general finite sequences.

6 Pretrained Model Evaluations

To measure how well language models solve state-tracking at differing levels of complexity without task-specific training, we also evaluate a suite of pretrained models on StateBench. We study two 7B parameter models: Olmo-3 (Team Olmo et al., 2025), a Transformer-NoPE model; and Olmo-3-Hybrid (Merrill et al., 2026b), a Hybrid$^{\pm}$ model, both pretrained, midtrained, and extended for long context on the Dolma dataset (Soldaini et al., 2024). We also study a family of smaller, custom pretrained models trained on the same dataset, at two size classes: 275M parameters (Transformer-NoPE, GDN$^{\pm}$, and Hybrid$^{\pm}$); and 2.7B parameters (Hybrid$^{\pm}$). Pretraining and midtraining data were capped at 8K tokens for all models, while long-context data was capped at 64K.

Since pre-trained models are frequently meant to be 'task-agnostic' tools, evaluating on one particular formatting may bias results or make them unreflective of general state-tracking performance. To account for this, we generate six different sequence formats for all evaluation samples, including integer code, spreadsheet cell filling, and cube painting (cf. Appendix C for details and examples). We include sequences of lengths up to 128K to test how well models can perform on sequences with lengths seen during pretraining (first vertical green line in the plots), seen only during long-context extension (second vertical green line), and longer than have been seen at all during training.

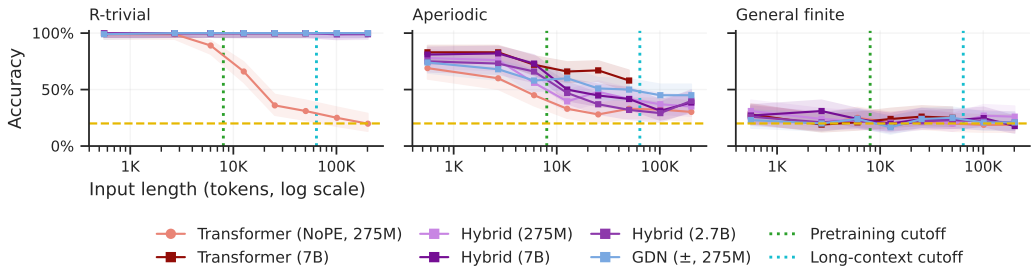

Figure 5: Pretrained models underperform theoretical expectations and direct-training results, showing consistent performance degradation on aperiodic data and completely failing on general finite data. Error bars show Wilson confidence intervals.

Pretrained models underperform at tracking state. We evaluate models on all six different sequence formats, and report in Figure 5 the maximum value obtained by the model on any of those

[8]For CruxEval samples, we take the codepath the function would take with the given input and unroll any loops; StateBench samples are already loop-free.

formats. Pretrained models consistently under-perform relative to theory and direct training. All models perform quite well on the r-trivial tasks except for the 275M-parameter transformer, which falls to at-chance accuracy beyond the pretraining length distribution. On aperiodic data, all models underperform expectations. The 7B models attain only ∼90% accuracy, and all models exhibit some degree of performance degradation within the pretraining length window, and noticeably drop afterwards. Some models, particularly the 275M GDN$^{\pm}$ model, manage to maintain accuracy of ∼50% beyond the long-context cutoff, but no higher. On general finite data, all models perform at-chance for all sequence lengths. In general, we find that *Krohn–Rhodes complexity*, *sample length*, and *model size* are the dominant factors impacting accuracy.

Models are sensitive to problem formatting. We find models exhibit a strong preference for some task formats over others (cf. Figures 9 and 10 in Appendix D). Data formatted like sequences from RULER (Hsieh et al., 2024) are much easier for models than the exact same sequences formatted like manipulations of named spreadsheet cells. This is especially true for small models, which attain near-ceiling performance on RULER-formatted r-trivial data but are near-chance on spreadsheet-formatted data, but even 7B-parameter models also generalize better to longer sequences when they are formatted like RULER tasks. RULER-formatted samples also have lower aggregate variance. This suggests that pretrained models are strongly affected by the particular choice of formatting for state-tracking tasks, and hence that evaluations of state-tracking capabilities in pretrained models should take care to measure performance across a variety of different formats.

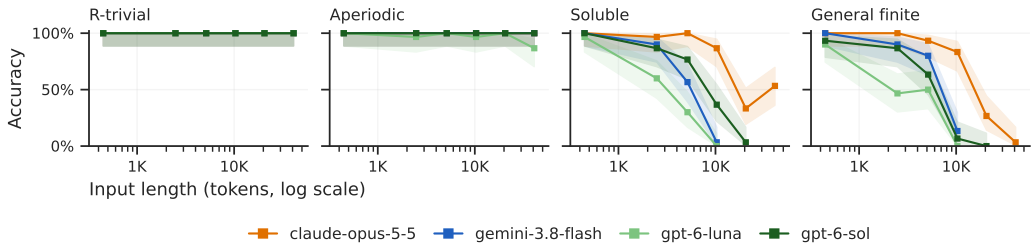

Figure 6: Frontier LLMs struggle on long-horizon soluble and general finite state-tracking tasks. Error bars show Wilson confidence intervals.

Frontier models struggle on long-horizon hard state tracking. We also run StateBench evaluations on four recently-released, proprietary frontier models: Gemini 3.8 Flash (Google, 2026), Claude Opus 5.5 (Anthropic, 2026), and GPT-6 Sol and Luna (OpenAI, 2026). We evaluate these models in a generative setting, where they are asked to produce the final state of the system instead of choose among 5 possible answers. As shown in Figure 6, all frontier models show strong performance on r-trivial and aperiodic problems; however, they fall to at-chance accuracy on soluble and finite state-tracking problems when inputs reach between 10K 64K tokens. The diminished performance of frontier reasoning LLMs shows that there is substantial headroom for improvement on generalizing on sensitive state-tracking problems.

7 Conclusion

We introduce a hierarchy of state-tracking based on the Krohn–Rhodes decomposition of finite monoids, classifying problems as r-trivial, aperiodic, soluble, and general. We place various models of Transformers and Recurrent models within this hierarchy, showing what kinds of state-tracking tasks they can express length-general solutions to. We validate these results by constructing StateBench, a dataset of state-tracking problems, and evaluate directly-trained, pretrained, and frontier language models on it, finding support for our theoretical results and highlighting the headroom left for language models to improve on state-tracking tasks. Several questions remain open. We do not know the exact placement of GDN^{+} with mixed or log precision; if it can implement modular counting then by Proposition 2 hybrid models with GDN^{+} sublayers would be able to solve all soluble tasks. Extending the hierarchy beyond finite monoids, i.e., to unboundedly-many variables, would bring many practical state-tracking tasks within the same framework, such as arbitrary discourse tracking. Empirically, our sensitivity analysis of code data suggests that general finite state tracking matters for code understanding, but where periodicity arises in natural data more broadly, and how much it drives the gaps between architectures in practice, remains an open question.

AI USE STATEMENT

In this work, we used generative AI tools for code generation, including to help write the library used to generate state-tracking data and to help write plotting code used to generate figures. We also used generative AI tools to offer critiques on drafts of this paper and offer phrasing suggestions. We have not used generative AI to generate evaluation data directly, nor to directly generate any figures present in this paper. We have reviewed all AI-assisted work through manual review. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For all formal results, we include detailed definitions and proofs in Appendices A and B. For all empirical results, we provide a more detailed overview in Appendices C and D. We will publicly release all data, code, and experimental setup conditions needed to reproduce our results in full.

REFERENCES

Alireza Amiri, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers, 4 February 2025. URL http://arxiv.org/abs/2502.02393.

Anthropic. Claude Opus 5.5. https://www.anthropic.com/claude-opus-5-5, 22 September 2026. Accessed: 2026-9-25.

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham M. Kakade, eran malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=8XWP2ewX-im.

David A Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, 1 October 1988. ISSN 0004-5411,1557-735X. doi: 10.1145/48014.63138. URL http://dx.doi.org/10.1145/48014.63138.

J. Richakd Büchi. Weak second-order arithmetic and finite automata. In Saunders Mac Lane and Dirk Siefkes (eds.), *The Collected Works of J. Richard Büchi*, pp. 398–424. Springer New York, New York, NY, 1990. ISBN 978-1-4613-8928-6. doi: 10.1007/978-1-4613-8928-6_22. URL https://doi.org/10.1007/978-1-4613-8928-6_22.

Google. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber. https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/, 2 September 2026. Accessed: 2026-9-25.

Riccardo Grazzi, Julien Siems, Jörg K.H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear RNNs through negative eigenvalues. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=UvTo3tVBk2.

J A Green. On the Structure of Semigroups. *Annals of Mathematics*, 54(1):163, 1 July 1951. ISSN 0003-486X,1939-8980. doi: 10.2307/1969317. URL http://dx.doi.org/10.2307/1969317.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. CRUXEval: A benchmark for Code Reasoning, Understanding and execution, 5 January 2024. URL http://arxiv.org/abs/2401.03065.

Michael Hahn and Mark Rofin. Why are Sensitive Functions Hard for Transformers?, 15 February 2024. URL http://arxiv.org/abs/2402.09963.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W Rae, Oriol Vinyals, and

Laurent Sifre. Training Compute-Optimal Large Language Models, 29 March 2022. URL http://arxiv.org/abs/2203.15556.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models?, 9 April 2024. URL http://arxiv.org/abs/2404.06654.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 977–996, Stroudsburg, PA, USA, 2025. Association for Computational Linguistics.

H Kamp. A Theory of Truth and Semantic Representation. In Jeroen A G Groenendijk (ed.), *Formal methods in the study of language*. U of Amsterdam, 1981.

Hans Kamp. *Tense Logic and the Theory of Linear Order*. Ph.d. dissertation, University of California, Los Angeles, 1968.

Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45 (6):983–1006, 1998. doi: 10.1145/293347.293351. URL https://doi.org.

Jeonghoon Kim, Byeongchan Lee, Cheonbok Park, Yeontaek Oh, Beomjun Kim, Taehwan Yoo, Seongjin Shin, Dongyoon Han, Jinwoo Shin, and Kang Min Yoo. Peri-LN: Revisiting normalization layer in the transformer architecture. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ci1S6wmXfO.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3835–3855, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.213. URL https://aclanthology.org/2023.acl-long.213/.

Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116(0): 450–464, 1965. ISSN 0002-9947,1088-6850. doi: 10.1090/s0002-9947-1965-0188316-1. URL http://dx.doi.org/10.1090/s0002-9947-1965-0188316-1.

Jiaoda Li and Ryan Cotterell. Characterizing the Expressivity of Fixed-Precision Transformer Language Models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=29LwAgLFpj.

Yanhong Li, Anej Svete, Ashish Sabharwal, and William Merrill. Efficiently representing algorithms with chain-of-thought transformers, 2026. URL https://arxiv.org/abs/2606.19697.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*. Number 65 in M.I.T. Research Monograph. The MIT Press, Cambridge, Mass., 1971.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Trans. Assoc. Comput. Linguist.*, 11:531–545, June 2023a.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought, 11 October 2023b. URL http://arxiv.org/abs/2310.07923.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers, 5 March 2025. URL http://arxiv.org/abs/2503.03961.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning*, 12 April 2024.

William Merrill, Hongjian Jiang, Yanhong Li, Anthony Lin, and Ashish Sabharwal. Why Are Linear RNNs More Parallelizable?, 2026a. URL http://arxiv.org/abs/2603.03612.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Chuan Li, Kyle Lo, Saumya Malik, D J Matusz, Benjamin Minixhofer, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. Olmo Hybrid: From theory to practice and back, 3 April 2026b. URL http://arxiv.org/abs/2604.03444.

William C. Merrill. *A Theory of the Computational Power and Limitations of Language Modeling Architectures*. PhD thesis, New York University, 2025. URL https://lambdaviking.com/assets/pdf/dissertation.pdf.

David A Mix Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. Regular languages in NC1. *Journal of Computer and System Sciences*, 44(3):478–499, 1 June 1992. ISSN 0022-0000,1090-2724. doi: 10.1016/0022-0000(92)90014-a. URL http://dx.doi.org/10.1016/0022-0000(92)90014-A.

Franz Nowak, Ryan Cotterell, and Reda Boumasmoud. An algebraic view of the expressivity of recurrent language models, 1 June 2026a. URL http://arxiv.org/abs/2606.01765.

Franz Nowak, Ryan Cotterell, and Reda Boumasmoud. A compositional theory of causally masked transformers, 29 July 2026b. URL http://arxiv.org/abs/2607.26988.

OpenAI. Introducing GPT-6 Sol and Luna. https://openai.com/index/introducing-gpt-6-sol-and-luna/, 23 September 2026. Accessed: 2026-9-25.

Jorge Pérez, P Barceló, and Javier Marinkovic. Attention is Turing-complete. *Journal of Machine Learning Research*, 22(75):75:1–75:35, 2021. ISSN 1532-4435,1533-7928. URL https://www.jmlr.org/papers/v22/20-302.html.

M P Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8 (2):190–194, 1 April 1965. ISSN 0019-9958,1878-2981. doi: 10.1016/s0019-9958(65)90108-7. URL http://dx.doi.org/10.1016/S0019-9958(65)90108-7.

Julien Siems, Riccardo Grazzi, Korbinian Pöppel, Kirill Kalinin, Hitesh Ballani, and Babak Rahmani. Learning state-tracking from code using linear RNNs, 16 February 2026. URL http://arxiv.org/abs/2602.14814.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: An open corpus of three trillion tokens for language model pretraining research, 31 January 2024. URL http://arxiv.org/abs/2402.00159.

Price Stiffler, Jr. Chapter 1. Extension of the fundamental theorem of finite semigroups. *Advances in Mathematics*, 11(2):159–209, 1 October 1973. ISSN 0001-8708,1090-2082. doi: 10.1016/0001-8708(73)90007-8. URL http://dx.doi.org/10.1016/0001-8708(73)90007-8.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding, 20 April 2021. URL http://arxiv.org/abs/2104.09864.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe

Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A Smith, and Hannaneh Hajishirzi. Olmo 3, 15 December 2025. URL http://dx.doi.org/10.48550/arXiv.2512.13961.

Pascal Tesson and Denis Therien. Logic meets algebra: The case of regular languages, 25 January 2007. URL http://dx.doi.org/10.48550/arXiv.cs/0701154.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 4 December 2017. Curran Associates Inc. ISBN 9781510860964.

Jie Wang, Tao Ji, Yuanbin Wu, Hang Yan, Tao Gui, Qi Zhang, Xuanjing Huang, and Xiaoling Wang. Length generalization of causal Transformers without position encoding, 18 April 2024. URL http://arxiv.org/abs/2404.12224.

Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in C-RASP: A transformer depth hierarchy, 19 June 2025. URL http://arxiv.org/abs/2506.16055.

Andy Yang, Blerta Veseli, Corentin Barloy, Michaël Cadilhac, Andreas Krebs, Charles Paperman, Howard Straubing, and Michael Hahn. Algebraic decomposition theory for transformer length generalization, 13 August 2026. URL http://arxiv.org/abs/2608.13433.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule, 9 December 2024. URL http://arxiv.org/abs/2412.06464.

A Formal Definitions

A.1 Monoids and Krohn–Rhodes

Definition 1 (Finite Monoid)**.** A *monoid* is a tuple $(M, \circ, 1)$ of a set M, an associative binary relation $\circ\colon M \times M \to M$, and an identity element 1 such that $1 \circ a = a \circ 1 = a$ for all $a \in M$. We say that a monoid is *finite* if M is finite.

Definition 2 (Word Problem)**.** Let M be a monoid, and let $w = w_1 \circ w_2 \circ \cdots \circ w_k$ be a *word* in M, i.e., a composition of finitely many elements $w_i \in M$. Since M is closed under $\circ$, this composition is in M as well. The *word problem* on M is to determine the identity of $w \in M$.

Definition 3 (Monoid Division)**.** For monoids A, B, we say that A *divides* B, written $A \mid B$, if there exists a submonoid $B' \leq B$ and a surjective homomorphism $B' \twoheadrightarrow A$.

Definition 4 (Wreath Product)**.** For finite monoids M, N their wreath product $M \wr N$ is the set $M^N \times N$ of pairs (f, n), where f is a map from N to M and $n \in N$, under the composition

$$(f, n)(g, n') \triangleq (h, nn'), \qquad \text{where } h(x) \triangleq f(x)\, g(xn) \text{ for all } x \in N,$$

with identity $(\mathbf{1}, 1)$, where $\mathbf{1}(x) \triangleq 1$ for all $x \in N$.

Intuitively, $M \wr N$ is a *cascade* in which N processes an input and M processes the input according to N's current state.

Definition 5 (Irreducibility)**.** A finite monoid M is irreducible if $M \mid (A \wr B)$ implies that $M \mid A$ or $M \mid B$ for all finite monoids A, B.

Theorem 3 (Krohn & Rhodes 1965)**.** *Let M be a finite monoid. Then there exist finitely-many irreducible factors $M_1, \ldots, M_k$ such that $M \mid (M_1 \wr M_2 \wr \ldots \wr M_k)$ and each M_i is either (a) the flip-flop monoid $U_2 \triangleq \{1, a, b\}$ where $xy = y$ for all $x, y \in \{a, b\}$; or (b) a finite simple group that divides M.*

Definition 6 (R-trivial Monoid)**.** A finite monoid M is *r-trivial* if $xM = yM$ implies $x = y$ for all $x, y \in M$.

Theorem 4 (Stiffler 1973)**.** *A finite monoid M is r-trivial if and only if $M \mid (U_1 \wr U_1 \wr \cdots \wr U_1)$ for some finite number of copies of U_1.*

Definition 7 (Aperiodic Monoid)**.** A finite monoid M is aperiodic if and only if $M \mid (U_2 \wr U_2 \wr \cdots \wr U_2)$ for some finite number of copies of U_2.

Definition 8 (Soluble Monoid)**.** A finite monoid M is soluble if and only if $M \mid (M_1 \wr M_2 \wr \cdots \wr M_k)$ for some finite k and each M_i is either (a) U_2 or (b) a cyclic group of prime order $\mathbb{Z}_p$ which itself divides M.

A.2 The Transformer Model

We adapt our formal model of transformers and RNNs from Merrill et al. (2026a); Merrill (2025). First, we define log- and polynomial-precision datatypes as in Merrill et al. (2026a, Section 2.1).

Let Σ be the token alphabet. We first define our notion of a multi-layer sequence model that will consume a sequence $w \in \Sigma^*$ and produce a next token:

Definition 9 (Multi-layer, multi-head sequence model; Merrill et al. 2026a; Merrill 2025)**.** A decoder-only transformer with h heads, d layers, model dimension m (divisible by h), and feed-forward width w is parameterized by

1. A fixed embedding matrix $\mathbf{E} : \mathbb{Q}^m \to \mathbb{Q}^{\|\Sigma\|}$;
2. For $1 \leq \ell \leq d$ and $1 \leq k \leq h$, query, key, and value projections $\mathbf{Q}_k^\ell, \mathbf{K}_k^\ell, \mathbf{V}_k^\ell : \mathbb{Q}^m \to \mathbb{Q}^{m/h}$. We also specify for each ℓ, k the head type: SMAT, AHAT, or GDN.
3. For $1 \leq \ell \leq d$, an output projection matrix $\mathbf{O}^\ell : \mathbb{Q}^m \to \mathbb{Q}^m$, initial feedforward projection $\mathbf{W}_1^\ell : \mathbb{Q}^m \to \mathbb{Q}^w$, and final feedforward projection $\mathbf{W}_2^\ell : \mathbb{Q}^w \to \mathbb{Q}^m$.
4. An output projection matrix $\mathbf{U} : \mathbb{Q}^m \to \mathbb{Q}^{\|\Sigma\|}$.

We now define how such a model computes a function $\Sigma^* \to \Sigma$.

Definition 10 (Merrill et al. 2026a; Merrill 2025)**.** Given an input $w \in \Sigma^*$, we apply inductive rules for each component, producing internal representations and, finally, an output as follows:

1. Embedding layer: Let $\mathbf{E}[w_i]$ is the embedding of w_i. For $1 \leq i \leq n$, we define
$$\mathbf{h}_i^0 = \mathsf{layer_norm}\left(\mathbf{E}[w_i]\right).$$
2. Self-attention sublayer: At each layer ℓ, we compute h attention heads. We first compute the sublayer input $\hat{\mathbf{h}}_i^{\ell-1} = \mathsf{layer_norm}(\mathbf{h}_i^{\ell-1})$. For each head $1 \leq k \leq h$, we compute queries $\mathbf{q}_{i,k}^\ell = \mathbf{Q}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$, keys $\mathbf{k}_{i,k}^\ell = \mathbf{K}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$, and values $\mathbf{v}_{i,k}^\ell = \mathbf{V}_k^\ell \hat{\mathbf{h}}_i^{\ell-1}$. The type of the head specifies a function that maps $\mathbf{q}_{i,k}^\ell, \mathbf{k}_{:,k}^\ell, \mathbf{v}_{:,k}^\ell$ to each head output $a_{i,k}^\ell \in \mathbb{Q}^{m/h}$ in a way we will return to defining. Finally, the self-attention sublayer then concatenates the output of h heads and then applies a linear projection to compute
$$\mathbf{g}_i^\ell = \mathbf{h}_i^{\ell-1} + \mathsf{layer_norm}\left(\langle \mathbf{a}_{i,1}^\ell, \ldots, \mathbf{a}_{i,h}^\ell \rangle \cdot \mathbf{O}^\ell\right).$$
3. Feedforward sublayer: Let $\hat{\mathbf{g}}_i^\ell = \mathsf{layer_norm}(\mathbf{g}_i^\ell)$. The feedforward sublayer computes the following tokenwise transformation:
$$\mathbf{h}_i^\ell = \mathbf{g}_i^\ell + \mathbf{W}_2^\ell(\mathbf{W}_1^\ell \hat{\mathbf{g}}_i^\ell)_+$$
4. Unembedding layer: The transformer outputs the token associated with $\arg\max \mathbf{h}_n^d \mathbf{U}$.

In this way, a multi-layer, multi-head sequence model $\mathcal{M}$ implements a function $\Sigma^* \to \Sigma$. To view such models as language recognizers, we can simply identify a token in Σ as a special "accept" symbol a and say that the transformer recognizes a language L if, for any string w, $w \in L \iff \mathcal{M}(w) = a$.

This architecture follows peri-norm (Kim et al., 2025): the use of layer-norm both before and after each sublayer (as well as on the embeddings). As discussed in Li et al. (2026, Appendix A.1), this is capable of simulating projected pre-norm (Merrill & Sabharwal, 2023b), a standard tool in AHAT constructions.

A.3 Sequence Mixers

The last step is to define the types of sequence mixers used by each architecture:

Definition 11 (SMAT Head)**.** The output of head at token i on queries $\mathbf{q}_i$, keys $\mathbf{k}_i$, and values $\mathbf{v}_i$ is

$$\mathbf{a}_i = \sum_{j=1}^{i} \frac{\exp(1/\tau \cdot \mathbf{q}_i^\top \mathbf{k}_j)}{Z_i} \cdot \mathbf{v}_j, \text{ where } Z_i = \sum_{j=1}^{i} \exp(1/\tau \cdot \mathbf{q}_i^\top \mathbf{k}_j)$$

and $\tau = \sqrt{d/h}$.

Definition 12 (AHAT Head)**.** The same as an SMAT head, but taking the limit $\tau \to 0$.

Definition 13 (GDN Head)**.** The output of head at token i is $\mathbf{a}_i$ defined via

$$\mathbf{S}_i^\ell = \alpha_i^\ell \big(\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top\big)\mathbf{S}_{i-1} + \beta_i \mathbf{k}_i \mathbf{v}_i^\top, \qquad \mathbf{a}_i = \mathbf{S}_i^\top \mathbf{q}_i,$$

with gate $\alpha_i \in [0, 1]$ and write strength β_i parameterized as in Yang et al. (2024). The eigenvalues of the transition $\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top$ are 1 and $1 - \beta_i$. Following Grazzi et al. (2025), we write GDN^+ when $\beta_i \in [0, 1]$ (nonnegative eigenvalues) and $\mathsf{GDN}^\pm$ when $\beta_i \in [0, 2]$ (eigenvalues in $[-1, 1]$).

By the classes $\mathsf{SMAT}, \mathsf{AHAT}, \mathsf{GDN}^+$, and $\mathsf{GDN}^\pm$, we denote the classes of languages expressible when all heads are the corresponding type.

B Expressivity Proofs

Lemma 1 (Implementing Basic Monoids)**.** *Various attention and recurrent mechanisms can solve various basic finite monoid word problems, as follows:*

a. There exists a one layer SMAT$_f$ that implements U_1.
b. There exists a one layer GDN$_f^+$ that implements U_2.
c. There exists a one layer AHAT$_\ell$ that implements U_2.
d. For any prime p, there exists an 8-layer AHAT$_\ell$ that implements $\mathbb{Z}_p$.

Proof of Lemma 1.a. To recognize word problems in the monoid $U_1 = \{1, 0\}$, we simply need to check if any 0 occurs in the context w. We do this by constructing an attention head h where the query is 1, the key j a boolean encoding whether token $w_j = 0$, and the value is 1. h will return 1 iff there exists some j such that $w_j = 0$. Thus, taking $1 - h$ will compute the word problem for U_1. □

Proof of Lemma 1.b. Ignoring the gating, GDN$_f^+$ has the form

$$S_t = (I - k_t k_t^\top) S_{t-1} + k_t v_t^\top.$$

Fixing a vector dimension d, let $\vec{i}$ be the one-hot vector for $i \leq d$. If $k_t = \vec{i}$ and $v_t = \vec{j}$, then the GDN update can be understood to replace the vector in row i of the state matrix with $\vec{j}$.

We now describe the updates required to implement U_2. For $w_t \in \{a, b\}$, let $k_t = \vec{1}$ and v_t be $\vec{1}$ or $\vec{2}$, respectively. Thus, these updates overwrite the first row of S_t to be either $\vec{2}$ or $\vec{3}$. For $w_t = 1$, let $\alpha_t = 0$, reducing the update to the identity. Thus, the first row if S_t is a representation of the U_2 word problem output. □

Proof of Lemma 1.c. To compute the U_2 word problem, we construct an AHAT attention head that attends over all tokens a, b in the context and retrieves the rightmost one. To do this, we add two terms to key j: the first term is a boolean indicator for whether $w_j \in \{a, b\}$, and the second is tie-breaking term in $(0, 1)$ that monotonically increases with j (Merrill & Sabharwal, 2023b, Appendix E). The query at i is just 1, and the value at j is simply w_j. Thus, at token i, the AHAT head returns the rightmost w_j such that $w_j \in \{a, b\}$, effectively solving the word problem. □

Proof of Lemma 1.d. Let representations of $z_1, z_2, \ldots z_n \in \{0, 1, \ldots, p-1\}$ be the input to a log-precision AHAT layer. To implement $\mathbb{Z}_p$, we would like it to compute $S_i \triangleq \left(\sum_{j=1}^{i} z_j\right) \bmod p$

at each token i. The "division lemma" of Merrill & Sabharwal (2025) is a valuable tool here: it shows how to compute $a \bmod p$ from a representation of a at position i using a few layers of AHAT. However, it assumes $a \leq i$, whereas our partial sum can be as large as $(p-1)\,i$. We get around this hurdle by leveraging the fact that p is a fixed constant, as follows.

First, for each $r \in \{0, 1, \ldots, p-1\}$, we construct a log-precision uniform attention head h_r that computes the representation of $c_r \triangleq |\{j \leq i \mid z_j = r\}|$, i.e., the *number* of input values z_j for $j \leq i$ that equal r. By definition, $c_r \leq i$. We can rewrite S_i in terms of c_r as follows:

$$\begin{aligned} S_i &= \left(\sum_{r=1}^{p-1} r\, c_r \right) \bmod p \\ &= \left(\sum_{r=1}^{p-1} r\, (c_r \bmod p) \right) \bmod p \end{aligned}$$

Since $c_r \leq i$, we can now apply the division lemma of Merrill & Sabharwal (2025) to the output of h_r in order to compute $c_r \bmod p$ for each r in parallel, using a total of 7 AHAT layers. An MLP layer can then multiply the result (which is at most $p-1$) with r, resulting in a representation of $b_r \triangleq r\,(c_r \bmod p)$, which is at most $(p-1)^2$. Another MLP layer can now compute $\sum_{r=1}^{p-1} b_r$, which is at most $(p-1)^3$. A final MLP layer can now apply a $\bmod p$ operation to this sum, resulting in S_i. We thus have an 8-layer log-precision AHAT that implements Z_p. □

Lemma 2 (Implementing Monoid Composition)**.** *Let M be a finite monoid with a KR decomposition $M_1 \wr M_2 \wr \cdots \wr M_k$. Suppose that for each i, an attention-based or recurrent architecture A_i of depth d_i can implement M_i. Let $d \triangleq \sum_{i=1}^{k} d_i$. Then the depth-$d$ composition $A_k \circ \ldots \circ A_2 \circ A_1$ of the architectures implements M.*

Proof. We prove this by induction over the number of composed monoids.

In the base case of $k = 0$, we simply have the input word w (equivalently, the identity applied to w) encoded in the initial layer.

For the inductive step, assume layer $\sum_{i=1}^{k-1} d_i$ has a representation of the partial products for input w for the monoid $M_1 \wr M_2 \wr \cdots \wr M_{k-1}$. We have a monoid M_k whose elements are from $f_k(M_1 \wr M_2 \wr \cdots \wr M_{k-1})$. We can first compute a representation of these elements using a feedforward network since f_k is just a finite lookup table of the already represented partial products. By assumption, our network can implement M_k with a block of d_k layers. Augmenting the network with this block of d_k layers, we see that layer $\sum_{i=1}^{k} d_i$ will have a representation of the partial products under $M_1 \wr M_2 \wr \cdots \wr M_k$. This completes the proof. □

Theorem 1 ($\mathsf{GDN}_f^+ = \mathsf{Aprd}$)**.** *There exists a fixed-precision GDN$^+$ that solves the word problem over M if and only if M is aperiodic.*

Proof. The upper bound follows from Grazzi et al. (2025), who showed that a fixed precision GDN$^+$ cannot express any periodic monoid. For the lower bound, by Proposition 1, M has a KR decomposition $M_1 \wr M_2 \wr \ldots \wr M_k$ such that every M_i is the flip-flop monoid U_2. By Lemma 1.b, U_2 can be implemented with a single GDN_f^+ layer. Applying Lemma 2, we obtain a k-layer GDN$^+$ that implements M. □

Theorem 2 ($\mathsf{AHAT}_\ell \supseteq \mathsf{Sol}$)**.** *If M is soluble, then there exists a log-precision AHAT that solves the word problem over M.*

Proof. By Proposition 1, M has a KR decomposition $M_1 \wr M_2 \wr \ldots \wr M_k$ such that every M_i is either the flip-flop monoid U_2 or $\mathbb{Z}_p$ for some prime p. By Lemma 1.c, U_2 can be implemented with a single AHAT_ℓ layer. By Lemma 1.d, Z_p can be implemented by a 8-layer AHAT_ℓ. Applying Lemma 2, we obtain that AHAT_ℓ can implement M. More specifically, if $M_1 \wr M_2 \wr \ldots \wr M_k$ contains $r \leq k$ groups of the form $\mathbb{Z}_p$, then there exists a $(k+7r)$-layer AHAT_ℓ that implements M. □

Proposition 2. *Hybrid* $(\mathsf{GDN}_f^+ : \mathsf{SMAT}_\ell : \mathsf{GM}) \supseteq \mathsf{Sol}$*, i.e., for any soluble* M*, there exists a hybrid model composed of fixed-precision GDN*$^+$*, log-precision SMAT, and mod gadgets* GM^p *for a fixed set of* p *that solves the word problem over* M*.*

Proof. By Proposition 1, M has a KR decomposition $M_1 \wr M_2 \wr \ldots \wr M_k$ such that every M_i is either the flip-flop monoid U_2 or the group $\mathbb{Z}_p$ for some prime p. By Lemma 1.b, U_2 can be implemented with a single fixed-precision GDN$^+$ layer. What remains to implement is $\mathbb{Z}_p$. Let representations of $z_1, z_2, \ldots, z_n \in \{0, 1, \ldots, p-1\}$ be the input to a single layer of log-precision SMAT. At token i, this layer can use uniform attention to compute a representation of the sum $y_i \triangleq \sum_{j=1}^{i} z_j$, e.g., $1/y_i$ or $\phi(y_i)$ or y_i/i. We can now feed this to the mod gadget GM^p to compute a representation of $z \bmod p$, thereby implementing $\mathbb{Z}_p$. Applying Lemma 2 shows that the full hybrid model can implement M. □

C Formatting Examples

Below are instances of the abstract general finite sample from Figure 2 rendered in the six different formats we use.

C.1 Integer Code

```
Track the current value of each variable after executing the assignments.

// ........
a <- 8;
c <- 10;
// ........
// ........
TMP <- 81;
b <- 11;
// ........
d <- 59;
TMP <- b;
// ........
a <- 38;
TMP <- a;
a <- c;
c <- TMP;
TMP <- d;
d <- b;
b <- TMP;

a ?= 9; b ?= 58; c ?= 37; d ?= 10; TMP ?= 58;
```

C.2 Spreadsheet Cells

```
Track the contents of each spreadsheet cell after following the instructions.

Enter 8 in cell A1.
.....................................
.....................................
.....................................
Enter 10 in cell C1.
.....................................
Enter 81 in scratch cell.
Enter 11 in cell B1.
Enter 59 in cell D1.
Copy the current contents of cell B1 into scratch cell.
Enter 38 in cell A1.
Copy the current contents of cell A1 into scratch cell.
Copy the current contents of cell C1 into cell A1.
.....................................
Copy the current contents of scratch cell into cell C1.
Copy the current contents of cell D1 into scratch cell.
Copy the current contents of cell B1 into cell D1.
Copy the current contents of scratch cell into cell B1.

Final cell contents:

cell A1 contains 10. cell B1 contains 59. cell C1 contains 38. cell D1 contains 11. scratch cell contains 59.
```

C.3 CUBE PAINTING

```
Determine the final color of every cube after following the instructions.

Paint cube 1 teal.
.............................
Paint cube 3 gray.
Paint mixing cube topaz.
Paint cube 2 maroon.
Paint cube 4 brick.
.............................
Paint mixing cube the same color as cube 2.
Paint cube 1 ivory.
Paint mixing cube the same color as cube 1.
Paint cube 1 the same color as cube 3.
Paint cube 3 the same color as mixing cube.
Paint mixing cube the same color as cube 4.
Paint cube 4 the same color as cube 2.
Paint cube 2 the same color as mixing cube.
.............................
.............................
.............................

Final colors:

cube 1 is gray. cube 2 is brick. cube 3 is ivory. cube 4 is maroon. mixing cube is brick.
```

C.4 PEOPLE IN ROOMS

```
Track where each person is after following the movements in order.

Alice enters the basement.
.............................
Carol enters the studio.
.............................
.............................
Taylor enters the toolshed.
Bob enters the lounge.
David enters the lighthouse.
Taylor moves to Bob's current location.
Alice enters the canteen.
Taylor moves to Alice's current location.
Alice moves to Carol's current location.
Carol moves to Taylor's current location.
Taylor moves to David's current location.
David moves to Bob's current location.
.............................
Bob moves to Taylor's current location.
.............................

Final locations:

Alice is in the studio. Bob is in the lighthouse. Carol is in the canteen. David is in the lounge. Taylor is
      in the lighthouse.
```

C.5 STATUS LIGHTS

```
Track the color displayed by each status light after following the instructions.

Set light A to teal.
Set light C to gray.
Set buffer light to topaz.
Set light B to maroon.
.....................................
Set light D to brick.
Set buffer light to the color currently displayed by light B.
Set light A to ivory.
Set buffer light to the color currently displayed by light A.
Set light A to the color currently displayed by light C.
.....................................
Set light C to the color currently displayed by buffer light.
.....................................
Set buffer light to the color currently displayed by light D.
.....................................
Set light D to the color currently displayed by light B.
Set light B to the color currently displayed by buffer light.
.....................................
```

```
Final light colors:

light A displays gray. light B displays brick. light C displays ivory. light D displays maroon. buffer light
      displays brick.
```

C.6 RULER

```
Some special magic numbers are hidden within the following text. Make sure to memorize it. I will quiz you
      about the numbers afterward.

One of the special magic numbers for hollow is: 9931280.
must have been when they first started to. It was presumably many
One of the special magic numbers for golden is: 3017169.
thousands of years between when people first started describing things as
One of the special magic numbers for temporary is: 8434201.
One of the special magic numbers for silver is: 5570834.
One of the special magic numbers for crimson is: 1829661.
The special magic number for temporary becomes the number currently assigned to silver.
hot or cold and when someone asked ''what is heat?'' No doubt it was a very
One of the special magic numbers for hollow is: 9580157.
The special magic number for temporary becomes the number currently assigned to hollow.
gradual process. We don't know if Plato or Aristotle were the first to
ask any of the questions they did. But their works are the oldest we have
The special magic number for hollow becomes the number currently assigned to golden.
The special magic number for golden becomes the number currently assigned to temporary.
The special magic number for temporary becomes the number currently assigned to crimson.
The special magic number for crimson becomes the number currently assigned to silver.
The special magic number for silver becomes the number currently assigned to temporary.

hollow is 3017169. silver is 1829661. golden is 9580157. crimson is 5570834. temporary is 1829661.
```

D Additional Statistics and Results

	Monoid Complexity Class			
	R-trivial	Aperiodic	Soluble	General finite
Max. dependency length (K)	0	0	21.3	32.1
Sensitivity	0.08 [0.00–0.91]	0.08 [0.00–0.91]	0.76 [0.06–1.00]	0.75 [0.04–1.00]
Prompt tokens (K)	1.16 [0.07–981]	1.16 [0.07–981]	1.34 [0.09–768]	1.25 [0.06–939]
Shared across classes				
Samples per condition	2,720			
Variables (m)	2, 4, 8, 16			
Assignments (n)	49 [11–64K]			

Table 1: Descriptive statistics for StateBench. Cells give medians, with [min, max] ranges in grey below. Token counts and dependency lengths are in thousands.

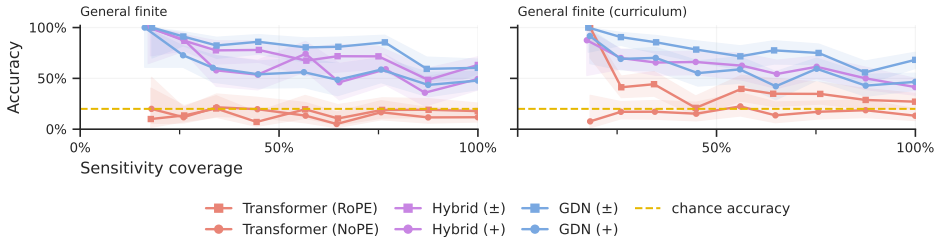

Figure 7: Effect of Sensitivity: On general finite state tracking data, direct trained models perform worse on higher sensitivity data. Error bars show Wilson confidence intervals.

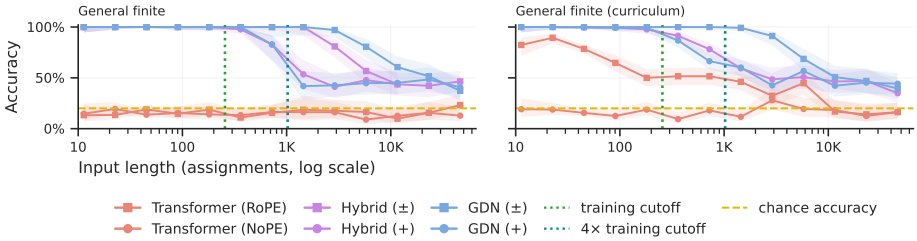

Figure 8: Curriculum Training: Sorting general finite sequences by sensitivity during training **(right)** boosts direct training performance of Transformer-RoPE models on in-distribution data compared to training on the same randomly-shuffled sequences **(left, repeated)**; Transformer-NoPE, GDN, and Hybrid models are unaffected. Error bars show Wilson confidence intervals.

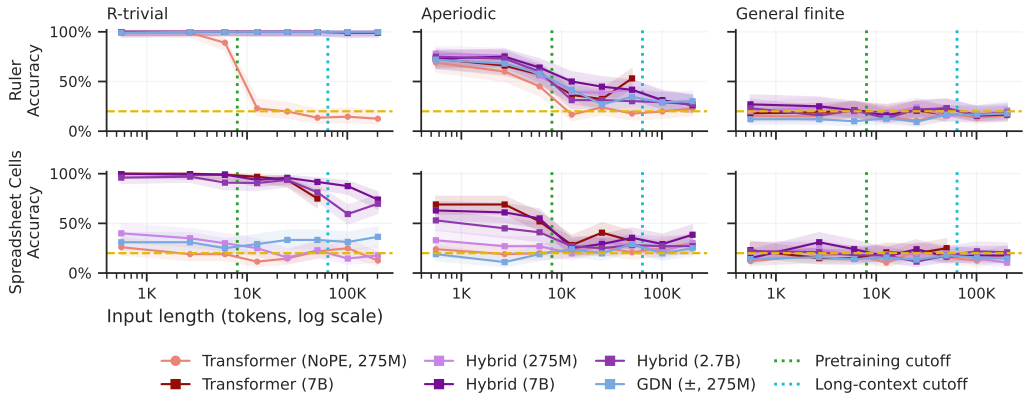

Figure 9: Pretrained base models perform better on RULER-formatted StateBench sequences **(top)** than on the same sequences formatted as spreadsheet manipulations **(bottom)**. Error bars show Wilson confidence intervals.

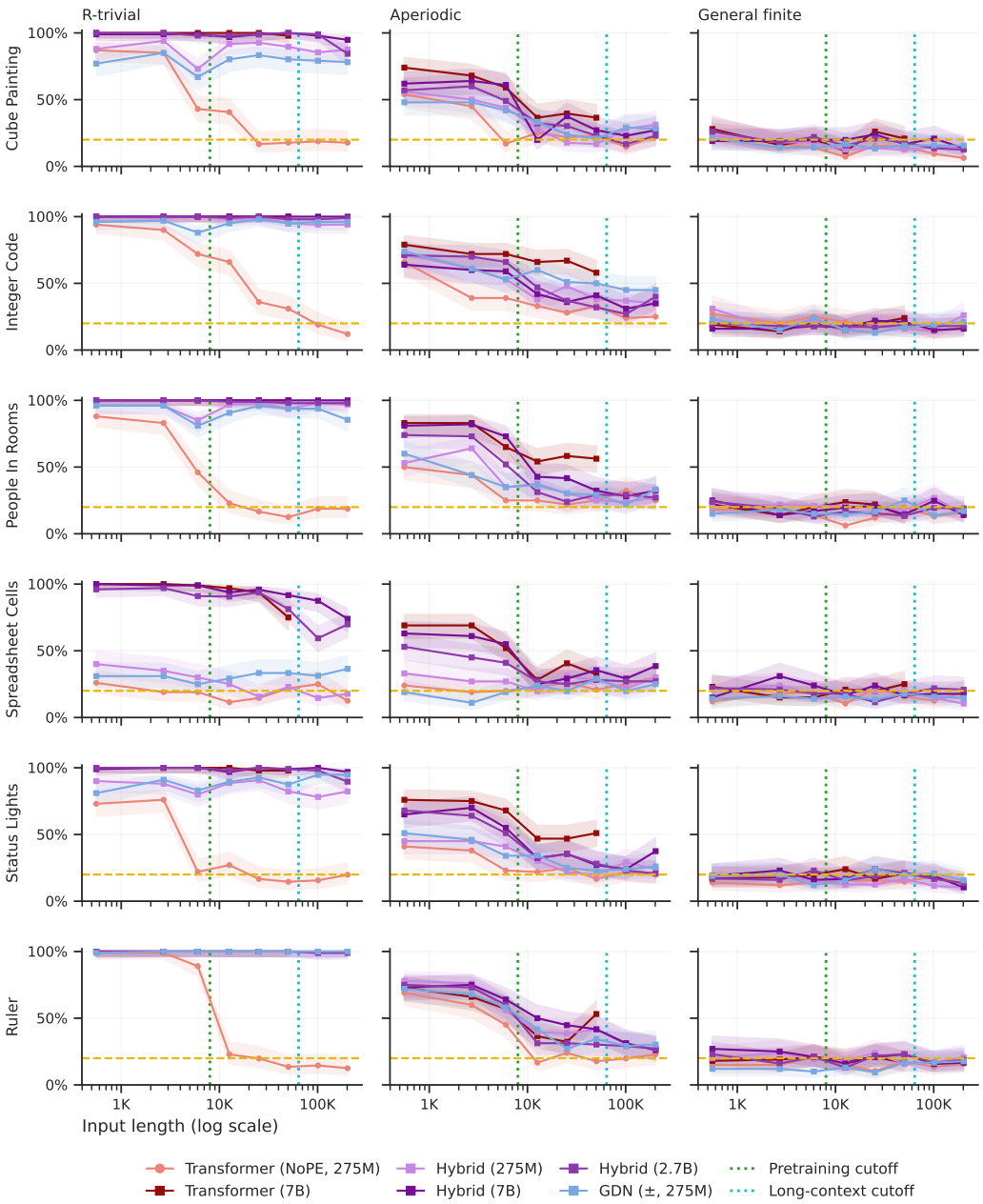

Figure 10: Pretrained models evaluated on StateBench problems by complexity class and formatting. Error bars show Wilson confidence intervals.