A UNIFIED FORMAL FRAMEWORK TO PREDICT LENGTH GENERALIZATION IN LINEAR RNNS, TRANSFORMERS AND HYBRID MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

C-RASP-based formal frameworks are currently the best known predictors of length generalization in Transformers. However, similar theoretical generalization guarantees do not yet exist for Linear RNNs (LRNNs) and Hybrid models (which combine LRNNs and Transformers), even though much work has characterized their capabilities empirically. To help explain these phenomena, we introduce a unified RASP variant, different slices of which characterize these models' length generalization behavior. We extend previous length generalization frameworks for Transformers to establish guarantees for both LRNNs and Hybrid models. Using this unified framework, we show that all these sequence models length generalize on different classes of tasks, but globally are constrained to sublinear communication complexity and therefore are unlikely to length generalize on copying, addition, and multiplication. We test our framework on standard task suites and find it to be predictive of empirical performance for all models.

1 INTRODUCTION

Modern LLMs are typically composed of Transformer layers (Vaswani et al., 2017), Linear RNN (LRNN) layers (Dao & Gu, 2024; Yang et al., 2025b, inter-alia) or, more recently, a mix of both (Merrill et al., 2026b; Kimi Team, 2026). These Hybrid models have been in part adopted because they combine complementary strengths of LRNN variants (Gu & Dao, 2024) and Transformers, which lie in distinct fragments of the complexity class $\mathbf{TC}^0$ (Merrill et al., 2024; Sarrof et al., 2024). However, through iterative improvements, recent LRNN variants (Yang et al., 2025b; Grazzi et al., 2025; Peng et al., 2025) allowing diagonal plus low rank (DPLR) transitions of their hidden state have been shown to be be able to solve $\mathbf{PNC}^1$-complete problems, which transformers cannot under standard complexity conjectures (Merrill et al., 2026a). This does not imply, however, that such LRNNs subsume transformers, and a more fine-grained analysis is required.

In principle, expressivity does not imply learnability, neither in the case of LRNNs (Back De Luca et al., 2025; Okpekpe & Orvieto, 2025; Shakerinava et al., 2026) nor in the case of Transformers (Hahn & Rofin, 2024; Huang et al., 2025a; Kraus et al., 2026). Thus, studies on learnability complement studies on expressivity. Of particular interest is learnability of length generalizing solutions, as modern LLMs are often subjected to context extensions after pretraining to improve performance on longer context windows (Press et al., 2022; Han et al., 2024; Jin et al., 2024; An et al., 2024).

C-RASP (Yang & Chiang, 2024; Huang et al., 2025a), a counting based logic, has so far turned out to be the most predictive tool to analyze length generalization in Transformers (Huang et al., 2025a; Jobanputra et al., 2025; Yang et al., 2025a; Chen et al., 2025a). However, analogous frameworks for reasoning about the generalization abilities of LRNNs are currently lacking — in particular, ones that compose with C-RASP to also cover Hybrid models. In this work, we take first steps towards filling this gap and make the following contributions:

• We introduce a unified RASP language, consisting of different slices which either recover known variants for Transformers (C-RASP[periodic, local] of Huang et al. (2025a), C*-RASP[local] of Sarrof et al. (2026)) or give us variants to reason about LRNNs (R-RASP) and hybrid models (H-RASP).

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.

• We define idealized learning frameworks similar to those of Huang et al. (2025a); Sarrof et al. (2026) for LRNNs and Hybrid models. We prove that, in these frameworks, the models are guaranteed to length generalize on tasks, for which we can write a program in the corresponding RASP variant. Through this lens, we identify lower bounds for both LRNNs and Hybrid models (See Figure 1).

• We establish upper bounds in these frameworks, showing that all such sequence models have sublinear communication complexity and hence are expected to not length generalize on tasks requiring linear communication like copying, multiplication and addition.

We test our framework on multiple tasks and find it to be predictive of when sequence models empirically length generalize, hence offering practitioners a unified tool to simultaneously reason about and compare model capabilities of Linear RNNs, Transformers and Hybrid models.

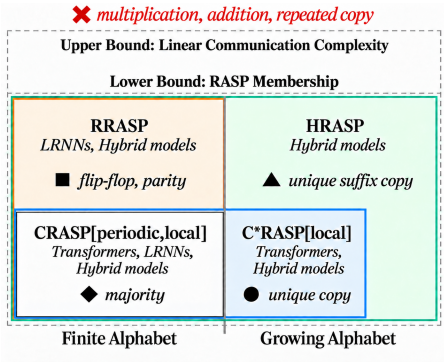


Figure 1: **Key results**

2 Background & Preliminaries

2.1 Sequence Model Architectures

Given input symbols $w_1, \ldots, w_n \in \Sigma$, an embedding E gives $x_t^{(0)} = E(w_t) \in \mathbb{R}^d$. For Linear RNNs, and thus also for Hybrid models, positional encodings are not required. We omit layer norm in accordance with past formalizations (Huang et al., 2025a; Sarrof et al., 2026), as it does not meaningfully affect the results. Each layer is as follows.

$$z_t^{(\ell)} = x_t^{(\ell)} + \mathrm{Head}^{(\ell)}\left(x_t^{(\ell)}\right)_t, \qquad x_t^{(\ell+1)} = z_t^{(\ell)} + \mathrm{FF}^{(\ell)}\left(z_t^{(\ell)}\right). \tag{1}$$

The Head function is always causal, with its output at t depending on positions $1 \ldots t$. For Transformer layers, this corresponds to a causal self-attention head with softmax, for LRNN layers this corresponds to some form of linear recurrence. All learned parameters have finite precision, intermediate computations can use exact arithmetic, but following Huang et al. (2025a) the output of the head needs to be rounded to fixed number of fractional bits. After the final layer, an unembedding matrix maps the last token representation to a logit distribution for the next token, $\ell_t = U(x_t^{(L)})$.

DPLR Linear RNNs: A *linear RNN* (LRNN) head maintains a state $S_t \in \mathbb{R}^{d_s \times d_v}$ where,

$$S_t = A_t S_{t-1} + B_t, \qquad A_t = A(x_t), \quad B_t = B(x_t), \quad y_t = S_t^\top q_t. \tag{2}$$

where $A_t, B_t \in \mathbb{R}^{d \times d}$ are functions of the current position's activation. We concatenate the outputs of H such heads for the output of Head in Equation 1. The update is *linear in the recurrent state*; the maps that produce A_t and B_t from x_t may themselves contain nonlinearities. The parameterization of A_t affects both the efficiency (Dao & Gu, 2024; Gu & Dao, 2024; Terzić et al., 2025; 2026b;a) and expressivity of SSMs (Merrill et al., 2024; Sarrof et al., 2024; Grazzi et al., 2025).

We focus specifically on variants that allow *diagonal-plus-low-rank* (DPLR) transitions of A_t, as they are maximally expressive among known LRNNs (Merrill et al., 2026a) and can solve tasks which Transformers cannot (in the constant depth classes $\mathbf{AC}^0$ (Hahn, 2020; Hao et al., 2022) or $\mathbf{TC}^0$ (Merrill et al., 2022; Chiang, 2025)). Besides, DPLR LRNNs are also of practical relevance, with many new LLM architectures employing them in their designs (Peng et al., 2025; Merrill et al., 2026b; Kimi Team, 2026). For simplicity we refer to DPLR LRNNs as LRNNs henceforth. We adopt the abstraction (described below in Equation 3) used by Merrill et al. (2026a) for theoretically analyzing these models, and show in Appendix A.2, how the specific variants in these LLMs are a special case of the abstraction below.

$$A_t = D_t - U_t V_t^\top, \qquad D_t \text{ diagonal}, \quad U_t, V_t \in \mathbb{R}^{d_s \times r}, \quad r = O(1). \tag{3}$$

Hybrid Models: Hybrid models interleave linear recurrence layers with causal softmax attention layers. We follow Huang et al. (2025a)'s parameterization of Transformers, using logarithmic scaling of attention scores (Chiang & Cholak, 2022) as well as allowing ReLU and Heaviside activations.

2.2 Theoretical Frameworks for Length Generalization in Transformers

C-RASP: Counting RASP is a counting logic introduced by Yang & Chiang (2024). Huang et al. (2025a) showed that in an idealized learning model, membership of a task to C-RASP or C-RASP[periodic,local] (an extension capturing positional information) guarantees whether a Transformer (without or with positional encodings respectively) length generalizes. While Huang et al. (2025a) considered Absolute and No Positional Encodings (APE and NoPE), more recently, the results of Yang et al. (2025a) together with those of Jerad et al. (2026) suggest that the more prevalent, Rotary Positional Encodings (RoPE) is subsumed by C-RASP[periodic, local], and thus the generalization guarantees might extend to RoPE as well. Despite the idealized learning regime under which it operates, C-RASP has since proved to be the most effective in predicting length generalization in Transformers (Huang et al., 2025a; Jobanputra et al., 2025; Yang et al., 2025a).

C-RASP is defined over a finite alphabet. However, some tasks such as Unique Copy, which involves simply copying over a sequence of distinct tokens, requires the alphabet size to grow with input length. Such tasks, where the underlying vocabulary or alphabet may change, but the algorithm for solving the task remains the same, therefore require reasoning over abstract symbols in some sense (Boix-Adserà et al., 2024). To formalize such cases, Sarrof et al. (2026) introduced C*-RASP[local], with minimal changes to the existing C-RASP and gave similar theoretical guarantees of length generalization for tasks where the underlying alphabet size grows with test length. This logic operates over $\Sigma \cup \mathcal{C}$, where Σ is finite, but $\mathcal{C} = \{c_1, c_2, \dots\}$ is a potentially infinite ordered set of symbols. To achieve such generalization practically, Sarrof et al. (2026); Kraus et al. (2026) used *offsets* not just for positions (as done to simulate longer lengths for training APE Transformers), but also for the subset of characters over which we want such generalization. Thus, c_k's in the input get shifted by a fixed offset o to get c_{k+o} where the offset depends on the maximum desired test length and vocabulary performance.

Idealized Learning: In Huang et al. (2025a) and Sarrof et al. (2026), generalization was formalized as asymptotically *converging* to the same underlying solution from a big pool of *possible hypotheses* (Transformers trained on increasingly longer context windows, that fit the training datasets perfectly) with a *regularizer* penalizing model complexity. These RASP variants served as more convenient one-way interfaces to these "algorithms", hence useful for establishing positive generalization guarantees in this idealized learning setup. These underlying algorithms were shown to be bounded by logarithmic communication complexity (thus Transformers were predicted to fail on tasks requiring more), under Yao (1983)'s model. In this model, the input is partitioned into two halves, and we calculate the minimum amount of bits needed to be transferred to the second half for successful computation of the output. In this paper, we will denote the hypothesis class and regularizer of Huang et al. (2025a) as $\Theta^{\mathrm{T}}, \Re^{\mathrm{T}}$ and that of Sarrof et al. (2026) as $\Theta^{\mathrm{T},*}, \Re^{\mathrm{T},*}$ respectively.

3 Theoretical Results

3.1 Hybrid RASP

Towards the goal of reasoning about Transformers, LRNNs and Hybrid models with the same framework, we introduce H-RASP, extending C-RASP. Taking different slices of H-RASP gives us corresponding frameworks for LRNNs or recreates the existing C-RASP framework for Transformers.

Definition 1 (H-RASP Programs)**.** *Consider a joint alphabet $\Omega = \Sigma \cup \mathcal{C}$, where Σ is finite and $\mathcal{C} = c_1, c_2, \dots$ is an ordered alphabet congruent with $\mathbb{N}$. A H-RASP program is a finite sequence of operations from Table 1. On an input $w = w_1 \dots w_n \in \Omega^*$, at each position, an operation can define either Boolean (P), integer (C), finite monoid (G) values. Each operation in the program can depend on any operation preceding it in the program sequence. Any constant, period, and monoid is fixed in the program independently of the input length. The program accepts w iff P is the final Boolean operation and $P(|w|) = \top$.*

Thus, in H-RASP and all of its slices, the value of the last Boolean valued operation at the last position of the input string determines if the program accepts the word or not.

The **Initial** predicate $Q_\sigma(i)$ checks if the current position contains a specific token (from the finite Σ). The **Counting** operation counts the number of past positions ($j \leq i$), where a specific boolean predicate was true, with an optional mask $\psi(i, j)$. The mask can either be unused ($\top$) or be set to a

Boolean-Valued Operations	
Initial	$P(i) := Q_\sigma(i)$ for $\sigma \in \Sigma$
Boolean	$P(i) := \neg P_1(i)$
	$P(i) := P_1(i) \wedge P_2(i)$
Constant	$P(i) := \top$
Comparison	$P(i) := [C_1(i) \leq C_2(i)]$
Periodic (-)	$P(i) := \phi(i)$
	where ϕ is periodic.
Membership	$P(i) := [G(i) \in B]$, $B \subseteq M$
	where M is a finite monoid.

Finite-Monoid Operations	
Prefix product	$G(i) := \Gamma_{M,\gamma}\langle P_1, \ldots, P_k\rangle(i)$
where M is finite and $\gamma : \{\top, \bot\}^k \to M$.	

Count-Valued Operations	
Counting	$C(i) := \#\,[j \leq i,\, \psi(i,j)]\;\; P(j)$
	either $\psi(i,j) := \top$ or
	$\psi(i,j) := i = j + \delta$
	δ is a local constant
Conditional	$C(i) := P(i)\ ?\ C_1(i)\ :\ C_2(i)$
Arithmetic	$C(i) := C_1(i) \pm C_2(i)$
Constant	$C(i) := 1$
Match (+)	$C(i) := \#\,[j \leq i, \chi(i,j)]\;\; P(j)$
	$\chi(i,j) := [\boldsymbol{c}_{j-\delta} = \boldsymbol{c}_{i-\gamma} + \tau]$
	δ, γ, τ are local constants
	$c \in \mathcal{C}$ (growing alphabet)

• H-RASP	Black + Blue + Orange
• R-RASP	Black + Orange
• C*-RASP$_{\text{atomic}}$[local]	Black + Blue − Periodic
• C-RASP[periodic, local]	Black

Table 1: **H-RASP operations.** The black operations give C-RASP[periodic, local] (Huang et al., 2025a). Adding **Membership** and **Prefix product** operations gives the syntax of R-RASP for LRNNs. We add the **Match** predicate to C-RASP[periodic, local] and remove **Periodic** to get the syntax of C*-RASP of Sarrof et al. (2026) that concerns symbolic generalization over a growing alphabet. All operations in the table are in H-RASP for Hybrid models.

fixed local offset ($i = j + 5$ lets us check if the given boolean predicate was true 5 positions before the current one). Conditional, arithmetic, boolean, and comparison operations are all pointwise operations. A **Periodic** operation ϕ has a fixed period $q \geq 1$, such that its value depends on $i \bmod q$. These so far constitute operations in C-RASP[periodic, local] of Huang et al. (2025a).

The **Match** predicate is the only operator over the growing alphabet $\mathcal{C}$ and can help match distant parts of an input *in-context*. The ordering within $\mathcal{C}$ allows matching for instance, tokens c_k to c_{k+1}, helping keep track of tokens that serve as index hints (Zhou et al., 2024; Kraus et al., 2026) to help disambiguate long parts of an input containing many repetitions. Adding Match and removing Periodic from C-RASP[periodic, local] leaves us with C*-RASP[local] of Sarrof et al. (2026)[1].

All *regular languages* are exactly the languages recognizable with finite monoids (Nerode & Sauer, 1957). It is known that not all regular languages are definable in C-RASP (Huang et al., 2025b), hence to be able to capture such languages we introduce some new operations. For a finite monoid $(M, \cdot, 1_M)$ and any map $\gamma : \{\top, \bot\}^k \to M$, the **Prefix product** is:

$$\Gamma_{M,\gamma}\langle P_1, \ldots, P_k\rangle(i) = \gamma(P_1(1), \ldots, P_k(1)) \cdots \gamma(P_1(i), \ldots, P_k(i)).$$

The product is taken in the input order and need not be commutative. The **Membership** operation tests whether this value belongs to a fixed subset $B \subseteq M$. Prefix product and Membership together with C-RASP[periodic, local] give us the slice of R-RASP, which as we will establish in Theorem 1 can be used to provide length generalization guarantees for LRNNs (therefore also Hybrid models).

Unlike for Transformers, we do not introduce two different logics for Hybrid models (finite alphabet vs. a growing one), and instead use the unified H-RASP with all the operations listed in Table 1. The usage or lack thereof of the match predicate automatically clarifies which regime we are in.

3.2 DPLR LRNNs

Length Generalization Guarantees on a Finite Alphabet: We start by defining a learning model for LRNNs analogous to the one in Huang et al. (2025a) for Transformers. Unlike Transformers,

[1]Sarrof et al. (2026) used additional activation functions in their parameterization of Transformers to get a more general form of the match predicate. As our goal is to have a unified framework, we instead use the one described in Table 1 which does *not* require any additional activations. We discuss the implications of this restriction on the results of Sarrof et al. (2026); Kraus et al. (2026) in Appendix A.1.4.

LRNNs process inputs of arbitrary length with a single fixed model, so we need not index the model family by context width. For an LRNN Π, let $\mathfrak{O}(\Pi)$ be the set of all of the learned parameter matrices and vectors (considered as one column matrices) in the model. This does not include the entire input dependent components of the recurrent head i.e. D_t, U_t, V_t, B_t, but does include the learned projections, biases, and convolutions used by the to produce these, besides including the embedding, unembedding, and MLP parameters. Let $p(\Pi)$ be the precision of the the model Π.

Definition 2 (Hypothesis Class for LRNNs)**.** *The hypothesis class Θ^R is the set of LRNNs (as defined in Eq 3) where each parameter vector and matrix is represented at p bits of precision for some $p \in \mathbb{N}$.*

Definition 3 (Regularizer LRNNs)**.** *Let $\Pi \in \Theta^R$. The regularizer $\mathfrak{R}^R(\Pi)$ is defined as follows.*

$$\mathfrak{R}^R(\Pi) := L + H + p(\Pi) + \sum_{W \in \mathfrak{O}(\Pi)} ||W||_F$$

Definition 4 (Inference Procedure)**.** *Given a function $f \in \mathcal{F}(\Sigma)$, let the subset of LRNN models from Θ^R that matches the behavior of f on all inputs of length $\leq \frac{n}{2}$ be represented as U_n^f. The Inference Procedure generates a sequence of LRNNs $\Pi_n \in U_n^f$ where*

$$\mathfrak{R}^R(\Pi_n) \leq \frac{1}{n} + \inf_{\Pi \in U_n^f} \mathfrak{R}^R(\Pi)$$

In our first result, we show that LRNNs are guaranteed to length generalize in our learning model, if the underlying task has a program in R-RASP.

Theorem 1. *(Idealized Length Generalization in the Limit for LRNNs) The inference procedure from Definition 4 applied to a target function $f \in \mathcal{F}(\Sigma)$ generates a sequence $\Pi_1, \Pi_2 \ldots$ with $\sup_{n=1,2,3} \mathfrak{R}(\Pi_n) < \infty$. If $f \in$ R-RASP, then there is some N_0 such that for all $m > N_0$, Π_m matches f on all inputs of any length $k \leq m$.*

Proof. The proof strategy here mirrors that of Huang et al. (2025a) and Sarrof et al. (2026), where a bounded regularizer results in a finitary characterization of our models, as bounding the Frobenius norm results in the number of non-zero entries in all the matrices of the model being bounded. Amongst the effectively distinct finite algorithms possible, either programs get ruled out at some input length n or match the behavior for all lengths, which has to happen after some finite N_0. All that remains to show is that R-RASP can be encoded in such finitary characterizations of LRNNs, which can be done by combining simple constructions from expressivity literature (Yang et al., 2026b). The full formal proof is given in Appendix B.1.1

□

Length Generalization Failure on a Growing Alphabet: We now follow Sarrof et al. (2026) in studying generalization in presence of a growing alphabet. This time, we do index LRNNs in our hypothesis classes. Thus, $\Pi \in \Theta_n^{\mathrm{R}}$ is an LRNN model that has an alphabet size n. Apart from this, the definition of our hypothesis class remains the same (Definition 2). We also retain the same regularizer $\mathfrak{R}^{\mathrm{R}}$ (Definition 3), but adjust our inference procedure to account for the growing alphabet.

Definition 5 (Modified Inference Procedure)**.** *Given $f : (\Sigma \cup \mathcal{C})^* \to \{0, 1\}$, let $U_n^{f,*}$ be the subset of LRNN models $\Pi \in \Theta^R$ that matches f on the restricted domain of inputs, with (i) length of input x, $|x| \leq n/2$ (ii) the tokens $c_k \in \mathcal{C}$ appearing in x span a range of $n/2$, i.e. if $K_x = \{k : c_k \in x\}$ is the set of indices of those character tokens, then $\max K_x - \min K_x \leq n/2$. Then choose $\Pi \in U_n^{f,*}$ such that $\mathfrak{R}^R(\Pi) \leq \frac{1}{n} + \inf_{\Pi \in U_n^{f,*}} \mathfrak{R}^R(\Pi)$.*

Theorem 2. *(Failure of LRNNs) There exist functions $f \in \mathcal{F}(\Sigma \cup \mathcal{C})$, where $f \in$ C*-RASP[local], and there is no sequence of LRNNs $\Pi_1, \Pi_2, \ldots$ with $\sup_{n \geq 1} \mathfrak{R}^R < \infty$ generated by the Inference procedure in Definition 5 such that after some $m > N_0$, Π_m matches f on inputs of any length $k \leq m$.*

Proof. The match predicate cannot be encoded in an LRNN in a way that works length generalizably, because while self-attention can compare symbols for exact matches in-context, LRNNs have to maintain all the information they have seen thus far to be ready to match any forthcoming token with its context. Thus, the memory required to solve the task needs to increase with increase in sequence length, but the the regularizer bounds the effective width. Full proof in Appendix B.1.2. □

This fact relates to empirical results that show LRNNs struggling on in-context recall on longer contexts (Arora et al., 2024; Okpekpe & Orvieto, 2025; Merrill et al., 2026b). Induction heads are canonically formed in trained Transformers and can solve such tasks (Elhage et al., 2021; Olsson et al., 2022), but our results suggest that such induction algorithms (that can be simulated via the match predicate, example construction in Appendix C.2) can't be learned generalizably by LRNNs.

3.3 Hybrid Models

By design, our learning model of LRNNs is compatible with Huang et al. (2025a) and with Sarrof et al. (2026) (upon restricting the match predicate). Thus one for Hybrid models follows naturally as a combination. For conciseness we informally state them here, and for self containedness, we recap Huang et al. (2025a)'s setup and give formal specifications for Hybrids in Appendix B.2.

Hypotheses. $\Theta^{\mathrm{H}}, \Theta^{\mathrm{H},*}$ consist of any possible interleaving of DPLR LRNN layers Θ^{R} with attention layers from (depending on whether we are choosing the setup for finite or growing alphabet) Θ^{T} of Huang et al. (2025a) or $\Theta^{\mathrm{T},*}$ of Sarrof et al. (2026). LRNNs carry implicit positional information (Kimi Team, 2025; Merrill et al., 2026b) and thus like for LRNNs, we do not need to index Hybrid models by context width.

Regularizers. $\Re^{\mathrm{H}}, \Re^{\mathrm{H},*}$ is obtained by summing the regularizers $\Re^{\mathrm{R}}$ with $\Re^{\mathrm{T}}$ or $\Re^{\mathrm{T},*}$. The common terms in both are only counted once (eg: number of layers, heads precision used etc.)

Inference Procedure. The Inference procedure are identical to Definition 4, 5 except that we pick models from Θ^{H} and $\Theta^{\mathrm{H},*}$ and use $\Re^{\mathrm{H}}, \Re^{\mathrm{H},*}$ to penalize complexity. We will refer to these inference procedures as $\mathcal{I}$ and $\mathcal{I}^*$ for finite and growing alphabet setting.

Corollary 1 (Generalization Guarantees for Hybrid Models)**.** *Let* $\Omega_m = \Sigma \cup \{c_1, c_2 \ldots, c_m\}$.

1. *If* $f \in$ *H-RASP, the inference procedure* $\mathcal{I}^*$ *applied to the target function* $f \in \mathcal{F}(\Omega)$ *generates* $\Pi_1, \Pi_2, \ldots$ *with* $\sup_{n\geq 1} \Re^{H,*}(\Pi_n) < \infty$*, then there is some* N_0 *such that for all* $m > N_0$*,* Π_m *matches* f *on all inputs over* Ω_m *of any length* $k \leq m$*, giving both length and symbolic generalization.*
2. *If* $f \in$ *R-RASP, the inference procedure* $\mathcal{I}$ *applied to the target function* $f \in \mathcal{F}(\Sigma)$ *generates* $\Pi_1, \Pi_2, \ldots$ *with* $\sup_{n\geq 1} \Re^{H}(\Pi_n) < \infty$*, then there is some* N_0 *such that for all* $m > N_0$*,* Π_m *matches* f *on all inputs over* Σ *of any length* $k \leq m$*.*

Corollary 1 follows from Theorem 1 and the generalization guarantees given for Transformers in Sarrof et al. (2026). While with a finite alphabet, Hybrid models share lower bounds with Linear RNNs, in the presence of a growing alphabet, they can perform iterated composition of C*-RASP[local], R-RASP operations, and thus more than the sum of their parts.

3.4 Upper Bounds for Length Generalization

Assume that Alice has access to the first N bits of an input word that is $2N$ bits long and Bob has access to the last N bits of the input. Huang et al. (2025a) had showed that Transformers cannot length generalize on tasks under their learning model if tasks required Alice to send more than $O(\log N)$ bits to Bob. Here, we show a slightly weaker statement for Hybrid models, which is that Hybrids (and thus also LRNNs) cannot length generalize on tasks that require Alice to send $O(N)$ bits to Bob. This entails that Hybrid models cannot generalize on copying, addition or multiplication.

Theorem 3. *(Communication Complexity Bottleneck) If a function* $f \in \mathcal{F}(\Sigma)$ *requires Alice to communicate* $\Omega(N)$ *bits to Bob, for Bob to correctly compute the output, then Hybrid models cannot length generalize on* f *under inference procedure* $\mathcal{I}$*.*

Proof. The communication between the halves of the input can either happen through attention, or by storing information in the hidden state. Huang et al. (2025a) showed that attention cannot transfer more than $\log(N)$ bits of information. Due to the finite characterization of LRNNs (Theorem 1), they also cannot store (thus not transmit) linear amount of information in its hidden state. □

4 EXPERIMENTS

Setup. We compare RoPE, NoPE, and APE Transformers with DPLR LRNNs, and Hybrid models across all our tasks. In our figures, we report the best performing Transformer accuracy across all the 3 PEs. We use OLMO-core's Gated DeltaNet implementation for LRNNs, its interleaved recurrent-attention models for Hybrids (Merrill et al., 2026b), except that we don't impose the $3:1$ ratio between the recurrence and attention layers. For Transformers with APE and NoPE we follow Huang et al. (2025a), with RoPE we follow OLMO-core again. Unless otherwise specified, consistent with prior work (Bhattamishra et al., 2020; Huang et al., 2025a; Jobanputra et al., 2025) we train only on positive strings on input lengths $[L_{min}, 50]$, (L_{min} is the shortest valid input length), and evaluate on bins $[L_{min}, 50], [51, 100], [101, 150]$ up to 3x the maximum train length. We describe all tasks and and their membership in our RASP variants in Appendix C. Accuracy is the fraction of test instances, where all tokens in an output string were correctly predicted by a model. We do a broad search to look for hyperparameters per task-architecture pair. We rerun the task pairs on the first good one for multiple seeds. Because our positive empirical results were sensitive to the random seed, we report the average performance over the top three seeds. To be fair in reporting negative empirical generalization, we do a more exhaustive hyperparameter search, going up to 8 layers. The best architecture is again evaluated on multiple seeds, out of which we report the average of the top three runs. We report all relevant training details in Appendix D.1 to reproduce our results.

Motivating example – Copying Variants. As a motivating example and to flesh out the differences between different architectures, we first start with presenting 4 kinds of copying tasks: *Unique* copy (copying a string where all tokens are distinct) and *Repeat* Copy (copying a string with no restrictions) from Huang et al. (2025a), *Selective* Copy from Cooper et al. (2026) and *Unique* Suffix Copy, a new variant we introduce. Cooper et al. (2026) defined Selective Copy over strings containing a fixed number of marker symbols ($m_1, \ldots m_k$) and normal tokens symbols (denoted by x_i). For the input word, the output of this task to retrieve the token at position $n - i$, where n is the length of the string, and i is the index of the largest marker symbol in the input string. This task as we will show is in R-RASP (Appendix C.3), but not in C-RASP[periodic, local], and hence only Linear RNNs and Hybrid models are supposed to generalize here. Finally in our new variant Unique Suffix Copy, we retain split marker tokens and normal tokens which also have to be of Selective Copy, but change the task to finding the *last* occurring marker symbol in-context (say m_i) and then outputting the suffix starting i tokens from the start. This task is in H-RASP, but not any of the smaller slices (Appendix C.2), and so only Hybrid models are expected to generalize. As Figure 2 shows our framework is able to predict precisely when length generalization happens in each of these models.

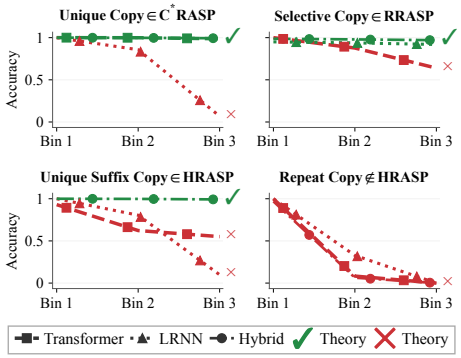


Figure 2: **Copying variants.** In each case, membership in the exact H-RASP slice is predictive of which models generalize.

Finer grained perspective. We next show how our framework is also useful in providing more fine grained perspectives on generalization performance of these models, reported in recent literature (Merrill et al., 2026a; Mohri, 2026).

1. Serialization in Addition: Mohri (2026) reported good length generalization in long-integer addition using their Hybrid model. However, in their setup, inputs to addition are fed digit by digit, with the least significant digit first. Such a task formulation has sublinear communication complexity and is in fact representable by WFAs (Droste et al., 2009), and is also in R-RASP (Appendix C.3), and thus we expect even LRNNs by themselves to length generalize. However if the input is fed with standard serialization $x + y = \text{sum}(x, y)$ the communication

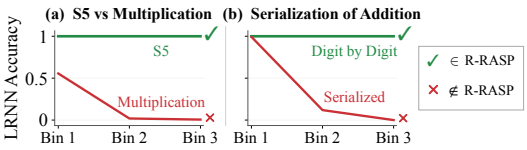


Figure 3: (a) $S5 \in \mathbf{NC}^1$, and Multiplication is in $\mathbf{TC}^0$, yet we see LRNNs generalizing only on the former. (b) Serialization in Addition matters, with even LRNNs generalizing on a digit by digit format, whereas standard serialization fails. Both times generalization track R-RASP predictions.

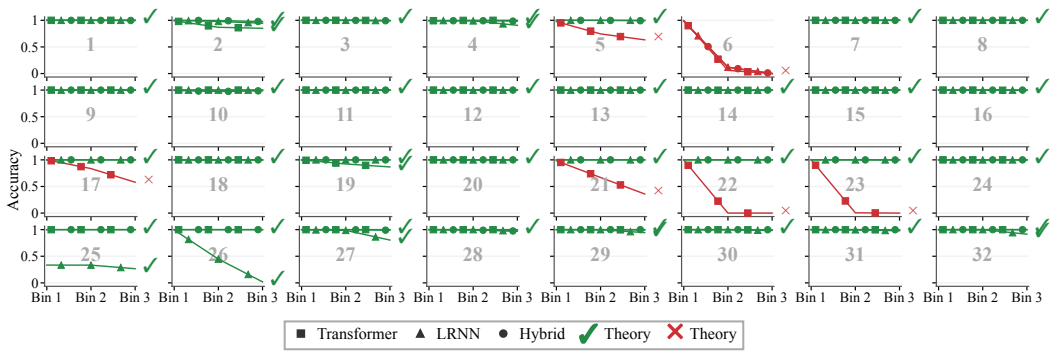


Figure 4: **Formal languages and algorithmic tasks.** We run all our models on the entire task suite of Huang et al. (2025a); Bhattamishra et al. (2020). The green lines (and a tick at the end) in the figure indicates when our framework predicts successful generalization, and red (with a cross) indicates when it predicts failure. Experiments strongly correlate with theory, with the only deviations for languages $a^n b^n c^n, a^n b^n c^n d^n$ for LRNNs. Full figure can be found in Appendix D.1, Figure 5.

complexity is linear, and thus our framework predicts LRNNs, Hybrids to not length generalize on these tasks, and thats what we also see in our experiments (See Figure 3b, and panel 6 in Figure 4).

2. Circuit Complexity: DPLR LRNNs can express solutions to $\mathbf{PNC}^1$-complete problems (Merrill et al., 2026a). We run experiments on Multiplication (in $\mathbf{TC}^0$) and on S_5 state tracking (an $\mathbf{NC}^1$-complete), where $\mathbf{TC}^0 \subseteq \mathbf{NC}^1 \subseteq \mathbf{PNC}^1$. Circuit complexity bounds might thus suggest that LRNNs and thus Hybrid models could perform well on both these languages. However, per our predictions, multiplication requires linear communication, whereas S_5, being a regular language, is $\in$ R-RASP. Thus, LRNNs and Hybrid models are expected to fail on multiplication, but succeed on S_5. Huang et al. (2025a) observed that while circuit complexity is useful for expressivity studies, it might be too coarse for predicting generalization in Transformers. We find the same to be true for LRNNs and Hybrids and instead find H-RASP to be helpful in giving fine grained predictions (See Figure 3a).

Formal Languages & Algorithmic Tasks. Finally, we run each of our models on the entire task suites of Bhattamishra et al. (2020); Huang et al. (2025a). These contain 17 regular, 9 counter and 6 algorithmic languages[2]. We follow the exact training setup of Huang et al. (2025a)) on regular and algorithmic, and that of Bhattamishra et al. (2020) on counter languages to help compare our results better. The entire counting suite is in C-RASP[periodic, local], and thus all models are expected to length generalize. As all regular languages are in R-RASP, our framework predicts both of these models to generalize on all the regular languages, but as Huang et al. (2025a) had showed these also include languages outside C-RASP[periodic, local], where Transformers are not expected to length generalize on all of them. Experiments largely track our theoretical predictions across most model, task pairs (See Fig 4). The exceptions are languages $a^n b^n c^n, a^n b^n c^n d^n$ (panel: 25, 26 in Fig 2), on which only LRNNs fail contrary to expectations. We suspect this to be because the N_0 required for unlocking generalization here might be much higher. Such cases highlight a key limitation predictions made by our framework: a guarantee that generalization will happen, but *not when*.

5 Discussion

Related Work: The introduction of RASP (Weiss et al., 2021) as an abstraction to help think about the algorithms inside Transformers sparked a wide range of work. Notably, Zhou et al. (2024) introduced RASP-L that restricted illegal indexing operations in RASP and empirically found that Transformers length generalized on problems with simple RASP-L programs. Huang et al. (2025a)'s idealized framework and use of C-RASP (Yang & Chiang, 2024) were in fact a formalization of this observation. While length generalization is also widely studied empirically (Anil et al., 2022; Press et al., 2022; Delétang et al., 2023; Kazemnejad et al., 2023, inter-alia), Huang et al. (2025a)

[2]Unique and Repeat Copy excluded, as they are already plotted in Figure 2

introduced a more formal, theoretically grounded way of reasoning about the ability of Transformers to generalize. Extensions have included, studying C-RASP with Chain of Thought (Jiang et al., 2026; Kraus et al., 2026), deriving bounds at which generalization happens (Chen et al., 2025a; Izzo et al., 2026), whether these bounds are computable (Yang et al., 2026a), finding fragments that are computable (Zetzsche et al., 2026). More RASP variants exist (Yang et al., 2024; Strobl et al., 2025; Huang et al., 2026), but they are orthogonal to studies concerning length generalization.

Studies focusing on the differences between Transformers and Linear RNNs have consistently highlighted the ability of Transformers to better perform recall (Arora et al., 2024; Jelassi et al., 2024; Wen et al., 2025; Bick et al., 2025; Ebrahimi et al., 2026) and for Linear RNNs to better keep track of state (Grazzi et al., 2025; Siems et al., 2025; 2026). Attempts at empirically understanding when Linear RNNs do not generalize include the *unexplored states hypothesis* (Buitrago Ruiz & Gu, 2025) that suggests longer test inputs might require state configurations that may not have been observed/formed during training. This has led to complementary attempts at improving context extension and recall in Linear RNNs (Chen et al., 2025b; Lu et al., 2025; Afzal et al., 2026, inter-alia). While such results increase the thresholds at which generalization starts to fail, underlying fundamental limitations in both Linear RNNs and Transformers persist, leading to practitioners opting for Hybrid models (Team, 2026; Mohri, 2026; Merrill et al., 2026b; Kimi Team, 2026).

Why an Idealized Learning Model: The idealized learning model used here for length generalization, extending similar results (Huang et al., 2025a; Jiang et al., 2026; Sarrof et al., 2026; Kraus et al., 2026) possesses the same flavor as risk minimization (Vapnik, 1998) and interpolation based on minimizing some complexity metric (Rissanen, 1978), which are all in some sense formalizations of Occam's Razor (favor simplicity) / Solomonoff induction (Solomonoff, 1964a;b).

These methods do not speak to actual gradient dynamics and it being predictive of downstream performance is purely correlational. Yet this correlation has been high across model types (Cortes & Vapnik, 1995; Schapire et al., 1998; Dziugaite & Roy, 2017; Jiang et al., 2019; Zhou et al., 2019; Lotfi et al., 2022; 2024a). Similarly for length generalization as well, this has proven to be the case (Huang et al., 2025a; Yang et al., 2025a; Jobanputra et al., 2025). While studying finer details about gradient dynamics are quite important, theoretical predictions of such idealized frameworks coupled with empirical verifications can help get more immediate actionable theoretical insights (Lotfi et al., 2024b; Kraus et al., 2026) into improving downstream empirical performance.

Amazing capabilities of LLMs: LLMs continue to improve with every iteration, with multiple open problems being solved continuously these days (Jin et al., 2026; OpenAI, 2026). In the space of such results, our negative results might look irrelevant. But solving such problems take enormous number of thinking steps and CoT tokens. Theoretically these models have been shown to be Turing complete (Pérez et al., 2019; Merrill & Sabharwal, 2024), but being able to do more in a single step is an obvious advantage in terms of cost and efficiency. Thus we believe that understanding the power of a single forward pass continues to be an important theoretical endeavor, and so far, multiple theoretical studies (Peng et al., 2025; Mohri, 2026; Cooper et al., 2026; Merrill et al., 2026b), including ours suggest that Hybrid models are a strictly better choice in this regard.

Limitations & Future Work: Our analysis is restricted to when the model answers immediately after the input, precluding common strategies to improve performance like Chain of Thought (Wei et al., 2022) or looping (Fan et al., 2025). For Transformers such extensions have been made (Jiang et al., 2026; Kraus et al., 2026), we leave extending the same to other models as future work. Strengthening our lower bounds especially for Hybrid models in the presence of a finite alphabet and extending our results to learning with gradient descent would also be exciting future work.

6 Conclusion

In conclusion, this work takes first steps towards providing a unified framework to reason, compare and predict length generalization capabilities of Transformers, LRNNs and Hybrid models. We empirically validate our theoretical predictions and see that they hold across a variety of tasks. C-RASP sparked a lot of work (e.g., inter alia Jobanputra et al., 2025; Chen et al., 2025a; Yang et al., 2025a; Huang et al., 2026; Kraus et al., 2026; Rizvi-Martel et al., 2026) to help deepen our understanding of the behavior of Transformers. We hope that our extension provides a similar useful lens for comparing architectures and helping guide future architecture design.

REFERENCES

Arshia Afzal, Aviv Bick, Eric P. Xing, Volkan Cevher, and Albert Gu. Raven: High-recall sequence modeling with sparse memory routing. *arXiv preprint arXiv:2607.25357*, 2026. URL https://arxiv.org/abs/2607.25357.

Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-free long-context scaling of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1493–1510. PMLR, 2024. URL https://proceedings.mlr.press/v235/an24b.html.

Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. 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=zSkYVeX7bC4.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LY3ukUANkO.

Artur Back De Luca, George Giapitzakis, Shenghao Yang, Petar Veličković, and Kimon Fountoulakis. Positional attention: Expressivity and learnability of algorithmic computation. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 2335–2390. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/back-de-luca25a.html.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7096–7116, 2020.

Aviv Bick, Eric P. Xing, and Albert Gu. Understanding the skill gap in recurrent models: The role of the gather-and-aggregate mechanism. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=hWYisuBbp7.

Enric Boix-Adserà, Omid Saremi, Emmanuel Abbe, Samy Bengio, Etai Littwin, and Joshua M. Susskind. When can transformers reason with abstract symbols? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=STUGfUz8ob.

Ricardo Buitrago Ruiz and Albert Gu. Understanding and improving length generalization in recurrent models. *arXiv preprint arXiv:2507.02782*, 2025. URL https://arxiv.org/abs/2507.02782.

Thomas Chen, Tengyu Ma, and Zhiyuan Li. Non-asymptotic length generalization. In *Forty-Second International Conference on Machine Learning (ICML)*, 2025a. URL https://openreview.net/forum?id=WZlq625BWD.

Yingfa Chen, Xinrong Zhang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. Stuffed mamba: Oversized states lead to the inability to forget. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=CdRauNXD1w.

David Chiang. Transformers in uniform TC^0. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=ZA7D4nQuQF.

David Chiang and Peter Cholak. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7654–7664, 2022.

John Cooper, Ilias Diakonikolas, Mingchen Ma, and Frederic Sala. Expressivity-efficiency tradeoffs for hybrid sequence models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=82EJxJzG6r.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 10041–10071. PMLR, 2024. URL https://proceedings.mlr.press/v235/dao24a.html.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=WbxHAzkeQcn.

Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2017. URL https://arxiv.org/abs/1703.11008.

Reza Ebrahimi, Michaël Defferrard, Sunny Panchal, and Roland Memisevic. On the "induction bias" in sequence models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZFDFVLjrka.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=PEdOdntGJG.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. doi: 10.1162/tacl_a_00306. URL https://aclanthology.org/2020.tacl-1.11/.

Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? In Lun Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14973–15008, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.800. URL https://aclanthology.org/2024.acl-long.800/.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3991–4008. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.222. URL https://aclanthology.org/2024.naacl-long.222/.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022. doi: 10.1162/tacl_a_00490. URL https://aclanthology.org/2022.tacl-1.46/.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A Formal Framework for Understanding Length Generalization in Transformers. In *Proceedings of ICLR*, 2025a. doi: 10.48550/arXiv.2410.02140. URL https://openreview.net/forum?id=U49N5V51rU.

Xinting Huang, Aleksandra Bakalova, Satwik Bhattamishra, William Merrill, and Michael Hahn. Discovering interpretable algorithms by decompiling transformers to RASP. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QnWjcTk14G.

Yu Huang, Zixin Wen, Aarti Singh, Yuejie Chi, and Yuxin Chen. Transformers provably learn chain-of-thought reasoning with length generalization. In *Proceedings of NeurIPS*, 2025b. URL https://arxiv.org/abs/2511.07378.

Zachary Izzo, Eshaan Nichani, and Jason D. Lee. Quantitative bounds for length generalization in transformers. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=TLSUIyBIfs.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=duRRoGeoQT.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Disentangling the expressivity of roPE. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=HBOZmG9MR9.

Hongjian Jiang, Michael Hahn, Georg Zetzsche, and Anthony Widjaja Lin. Softmax transformers are turing-complete. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FdkPOHlChS.

Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJlQfnCqKX.

Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. LLM maybe LongLM: SelfExtend LLM context window without tuning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22099–22114. PMLR, 2024. URL https://proceedings.mlr.press/v235/jin24b.html.

Jiashun Jin, Zheng Tracy Ke, and Bingcheng Sui. The gold rush in ai4math: Where are we now?, 2026. URL https://arxiv.org/abs/2608.24961.

Mayank Jobanputra, Yana Veitsman, Yash Sarrof, Aleksandra Bakalova, Vera Demberg, Ellie Pavlick, and Michael Hahn. Born a transformer – always a transformer? on the effect of pretraining on architectural abilities. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Huw15LqglI.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=Drrl2gcjzl.

Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025. URL https://arxiv.org/abs/2510.26692.

Kimi Team. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL https://arxiv.org/abs/2607.24653.

Oliver Kraus, Yash Sarrof, Yuekun Yao, Alexander Koller, and Michael Hahn. Barriers to universal reasoning with transformers (and how to overcome them). In *Conference on Language Modeling*, 2026.

Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew Gordon Wilson. PAC-bayes compression bounds so tight that they can explain generalization. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/cbeec55c50c3367024bafab2438a021b-Abstract-Conference.html.

Sanae Lotfi, Marc Anton Finzi, Yilun Kuang, Tim G. J. Rudner, Micah Goldblum, and Andrew Gordon Wilson. Non-vacuous generalization bounds for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32801–32818. PMLR, 2024a. URL https://proceedings.mlr.press/v235/lotfi24a.html.

Sanae Lotfi, Yilun Kuang, Brandon Amos, Micah Goldblum, Marc Finzi, and Andrew Gordon Wilson. Unlocking tokens as data points for generalization bounds on larger language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/11715d433f6f8b9106baae0df023deb3-Abstract-Conference.html.

Peng Lu, Jerry Huang, Qiuhao Zeng, Xinyu Wang, Boxing Wang, Philippe Langlais, and Yufei Cui. Mamba modulation: On the length generalization of mamba. *arXiv preprint arXiv:2509.19633*, 2025. URL https://arxiv.org/abs/2509.19633.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=NjNGlPh8Wh.

William Merrill, Ashish Sabharwal, and Noah A. Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022. doi: 10.1162/tacl_a_00493. URL https://aclanthology.org/2022.tacl-1.49/.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35492–35506. PMLR, 2024. URL https://proceedings.mlr.press/v235/merrill24a.html.

William Merrill, Hongjian Jiang, Yanhong Li, Anthony Widjaja Lin, and Ashish Sabharwal. Why are linear RNNs more parallelizable? In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=29sn1uqWn3.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, Nathan Lambert, Jacob Morrison, Luca Soldaini, Finbarr Timbers, Pete Walsh, Noah A. Smith, Hannaneh Hajishirzi, and Ashish Sabharwal. Olmo hybrid: From theory to practice and back. *arXiv preprint arXiv:2604.03444*, 2026b. URL https://arxiv.org/abs/2604.03444.

Mehryar Mohri. Rational transductors. *arXiv preprint arXiv:2602.07599*, 2026. URL https://arxiv.org/abs/2602.07599.

A Nerode and BP Sauer. Fundamental concepts in the theory of systems. Technical report, Technical Report 57-624, Wright Air Development Center, 1957.

Destiny Okpekpe and Antonio Orvieto. Revisiting associative recall in modern recurrent models. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025. URL https://openreview.net/forum?id=CcqAd5RPk5.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

OpenAI. Finite time blowup for navier–stokes, 2026. URL https://cdn.openai.com/pdf/32d9f210-8b73-45e0-91bc-82a30aef8a9a/navier-stokes.pdf.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "goose" with expressive dynamic state evolution. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ayB1PACN5j.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGBdo0qFm.

J. Rissanen. Paper: Modeling by shortest data description. *Automatica*, 14(5):465–471, September 1978. ISSN 0005-1098. doi: 10.1016/0005-1098(78)90005-5. URL https://doi.org/10.1016/0005-1098(78)90005-5.

Michael Rizvi-Martel, Satwik Bhattamishra, Guillaume Rabusseau, and Michael Hahn. From expressivity to sample complexity: Narrow teachers for transformers via c-rasp. *arXiv preprint arXiv:2607.11760*, 2026.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The expressive capacity of state space models: A formal language perspective. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2405.17394.

Yash Sarrof, Yupei Du, Katharina Stein, Alexander Koller, Sylvie Thiebaux, and Michael Hahn. On the ability of transformers to verify plans. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=KmOje8ebAg.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998. doi: 10.1214/aos/1024691352. URL https://www.schapire.net/papers/SchapireFrBaLe98.pdf.

Mehran Shakerinava, Behnoush Khavari, Siamak Ravanbakhsh, and Sarath Chandar. The expressive limits of diagonal SSMs for state-tracking. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5bg5Ru5OML.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear RNNs via householder products. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://openreview.net/forum?id=SoRiaijTGr.

Julien Siems, Riccardo Grazzi, Kirill Kalinin, Hitesh Ballani, and Babak Rahmani. Learning state-tracking from code: Repl traces and probabilistic automata. In *Workshop on Latent {\&} Implicit Thinking {\textendash} Going Beyond CoT Reasoning*, 2026.

Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1): 1–22, 1964a.

Ray J Solomonoff. A formal theory of inductive inference. part ii. *Information and control*, 7(2): 224–254, 1964b.

Lena Strobl, Dana Angluin, David Chiang, Jonathan Rawski, and Ashish Sabharwal. Transformers as transducers. *Transactions of the Association for Computational Linguistics*, 13:200–219, 2025. doi: 10.1162/tacl_a_00736. URL https://aclanthology.org/2025.tacl-1.9/.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Aleksandar Terzić, Michael Hersche, Giacomo Camposampiero, Thomas Hofmann, Abu Sebastian, and Abbas Rahimi. On the expressiveness and length generalization of selective state space models on regular languages. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20876–20884, 2025.

Aleksandar Terzić, Francesco Carzaniga, Nicolas Menet, Yannick Biehl, Michael Hersche, Thomas Hofmann, and Abbas Rahimi. Flash pd-ssm: Memory-optimized structured sparse state-space models. *arXiv preprint arXiv:2605.19150*, 2026a.

Aleksandar Terzić, Nicolas Menet, Michael Hersche, Thomas Hofmann, and Abbas Rahimi. Structured sparse transition matrices to enable state tracking in state-space models. *Advances in Neural Information Processing Systems*, 38:83072–83111, 2026b.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. 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=_VjQlMeSB_J.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11080–11090. PMLR, 2021. URL https://proceedings.mlr.press/v139/weiss21a.html.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. In *International Conference on Learning Representations*, volume 2025, pp. 48813–48856, 2025.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=FmhPg4UJ9K.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/13d7f172259b11b230cc5da8768abc5f-Abstract-Conference.html.

Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in C-RASP: A transformer depth hierarchy. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=jPduiyxyfw.

Andy Yang, Pascal Bergsträßer, Georg Zetzsche, David Chiang, and Anthony W. Lin. Length generalization bounds for transformers. *arXiv preprint arXiv:2603.02238*, 2026a. URL https://arxiv.org/abs/2603.02238.

Andy Yang, Christopher Watson, Anton Xue, Satwik Bhattamishra, Jose Llarena, William Merrill, Emile Dos Santos Ferreira, Anej Svete, and David Chiang. The transformer cookbook. *Transactions on Machine Learning Research*, 2026b. ISSN 2835-8856. URL https://openreview.net/forum?id=sPshCSvDrX.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=r8H7xhYPwz.

Andrew C Yao. Lower bounds by probabilistic arguments. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pp. 420–428. IEEE, 1983.

Georg Zetzsche, Hongjian Jiang, Andy Yang, Pascal Bergsträßer, Marco Sälzer, David Chiang, and Anthony W Lin. Length generalization for transformers via compression. *arXiv preprint arXiv:2609.08851*, 2026.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AssIuHnmHX.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: A PAC-bayesian compression approach. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1804.05862.

CONTENTS

1 Introduction **1**

2 Background & Preliminaries **2**

2.1 Sequence Model Architectures 2

2.2 Theoretical Frameworks for Length Generalization in Transformers 3

3 Theoretical Results **3**

3.1 Hybrid RASP 3

3.2 DPLR LRNNs 4

3.3 Hybrid Models 6

3.4 Upper Bounds for Length Generalization 6

4 Experiments **7**

5 Discussion **8**

6 Conclusion **9**

A Additional Background **17**

A.1 Learning Framework for Transformers 17

A.1.1 Common Notation and Notions 17

A.1.2 Finite Alphabet 18

A.1.3 Growing Alphabet 19

A.1.4 Implications of removing the reciprocal activation function 20

A.2 DPLR Linear RNNs 20

B Main Proofs **23**

B.1 Length Generalization in DPLR LRNNs 23

B.1.1 Finite Alphabet 23

B.1.2 Growing Alphabet 25
B.2 Hybrid Models 25
B.2.1 Finite Alphabet 26
B.2.2 Growing Alphabet 26

C Task Descriptions and RASP expressiveness **26**
C.1 Regular and Counter Languages considered 27
C.1.1 Regular Languages 27
C.1.2 Counter Languages 29
C.2 Algorithmic Tasks 31
C.3 Additional Tasks considered 33

D Additional Experimental Details **35**
D.1 Details on Training Setup 35

E AI Usage **42**

A Additional Background

A.1 Learning Framework for Transformers

For self containedness, we recap the learning frameworks of Transformers proposed by Huang et al. (2025a) (finite alphabet) and Sarrof et al. (2026) (growing alphabet).

A.1.1 Common Notation and Notions

Parameters and computation. A Transformer T has context width $N(T)$ (i.e. maximum allowed input size), width d, number of layers L, number of heads per layer H. The embedding and positional dimensions are denoted by $E_\sigma, p_i \in \mathbb{R}^d$, where σ will be a character from a given alphabet. The key, query and value are denoted layer and headwise as $K_{l,h}, Q_{l,h}, V_{l,h}$. Since we use A_t, B_t to refer to the parts of the hidden state in Equation 2, we shift away from Huang et al. (2025a) and Sarrof et al. (2026)'s notation for MLP parameters and call it C_l, D_l, b_l. We use $||.||_{op}, ||.||_F, ||.||_2$ for spectral, frobenius and euclidean norm respectively.

Positional offsets are added to simulate longer test length, $0 \leq o_p \leq N(T) - |x|$, where x is the input word. A similar offset o_c is also added to simulate character embeddings not encountered naturally during training upto the maximum desired alphabet size. Thus, at the first layer, we get $y_i^{(0)} = E_{x_i} + p_{i+o_p}$, where $|x| + o \leq N(T)$. The attention score from key position $j \leq i$ to query position i is $(y_j^{(\ell-1)})^\top K_{\ell,h}^\top Q_{\ell,h} y_i^{(\ell-1)}$.

Therefore attention logits, at query position i and key position j are computed as

$$a_{i,j}^{(l,h)} = (y_j^{(l-1)})^\top K_{l,h}^\top Q_{l,h} y_i^{(l-1)} \text{ for } 1 \leq j \leq i \leq |x|;\ l = 1, \ldots, L;\ h = 1, \ldots, H \tag{4}$$

Softmax attention has $\log |x|$ scaling in line with previous theoretical work suggesting its utility (Chiang & Cholak, 2022).

$$Y_i^{(l)} := y_i^{(l-1)} + \sum_{h=1}^{H} \frac{\sum_{j=1}^{i} \exp\left(\log |x| \cdot a_{i,j}^{(l,h)}\right) V_{l,h} y_j^{(l-1)}}{\sum_{j=1}^{i} \exp\left(\log |x| \cdot a_{i,j}^{(l,h)}\right)} \tag{5}$$

After each attention block, the activations are passed through a one-layer MLP:

$$y_i^{(l)} := Y_i^{(l)} + D_l \cdot f_l^{\text{act}}(C_l Y_i^{(l)} + b_l) \tag{6}$$

Both frameworks omit layer normalization and allow f^{act} activation to be ReLU and Heaviside functions. Sarrof et al. (2026, Appendix C.1) additionally allows a reciprocal activation. This is crucial, and for homogenizing the results across all frameworks, we will only utilize a restricted subset of results given in Sarrof et al. (2026) to avoid this non standard activation function. We discuss the implications on the resultant C*-RASP[local] framework in Section A.1.4.

Both frameworks assume finite precision for the parameters, attention scores are rounded to a fixed number of fractional bits before processing as well. However other any activations are allowed exact arithmetic.

Product Functions. A key idea behind the generalization results in these frameworks was the reparameterization of Transformations in terms of scalar product functions. These functions can be thought of as activations which have been decomposed into sums of different products, where each product is an inner products of the learnable parameters with the input token and/or position. As an example, consider two inputs $E_x + p_i$, $E_y + p_j$. When these interact through an attention layer, we can view the output score as being composed of the sums like $E_xKQE_x, E_xKQp_j, p_jKQE_y, p_iKQE_j$. Such a parameterization was crucial, as the number of these product functions is independent of the width used by the Transformer, thus allowing the framework to talk about increasing width sizes, and yet being able to control the kind of underlying algorithms that may appear in the model. This was done through two kinds of constraints.

1. *Translation Invariance*: To ensure that the output of a model is the same irrespective of whether the input starts at position 1 or at some offset o_p, the output of the model was asked to be translation invariant. This meant that product functions which involved only one positional embedding had the same value irrespective of which positional embedding was found in such products, whereas the product functions involving two positions had to have the same contribution, even if they were both shifted by some offset, i.e.
$$p_i^\top M p_j = p_{i+\Delta}^\top M p_{j+\Delta} \quad \text{whenever } 1 \le i, j, i+\Delta, j+\Delta \le N(T)$$
While Huang et al. (2025a) only imposed this on positions, Sarrof et al. (2026) imposed it on the growing part of the alphabet to ensure learning of the underlying abstract rule rather than a memorized rule based on some absolute identity.
2. *Locality*: This simply meant that interactions had to be local, and far away positional interactions or interactions between characters with distant embeddings (only characters part of the $\mathcal{C}$ alphabet, and not Σ) were discouraged. Thus if positions i, j were interacting, then when $i - j > \delta$ (where δ is a constant), the product function would just be 0.

Common Terms in the Regularizer. While the regularizer between the two settings of alphabet had slightly differing definitions, there were many common terms. To avoid duplication in definitions, we denote the sum of those common terms as $\mathcal{B}(T)$, where

$$\begin{aligned}\mathcal{B}(T) := L + H + \rho(T) + \max_{\ell,h} \|K_{\ell,h}^\top Q_{\ell,h}\|_{\text{op}} + \max_{\ell,h} \|V_{\ell,h}\|_{\text{op}} \\ + \max_{\ell} \|C_\ell\|_{\text{F}} + \max_{\ell} \|D_\ell\|_{\text{F}} + \|U\|_{\text{op}} \\ + \max_{1 \le i \le N(T)} \|p_i\|_2 + \max_{\ell} \|b_\ell\|_2. \end{aligned} \tag{7}$$

A.1.2 Finite Alphabet

We now finally present the learning models defined in Huang et al. (2025a) over a finite alphabet Σ, rewording them and adjusting the definitions based on the notation defined in this paper.

Definition 6 (Transformer hypothesis class (Huang et al., 2025a, Definition 4))**.** *For each $n = 1, 2, 3 \ldots$, the hypothesis class Θ_n^T is the set of Transformers T, with (1)$N(T) = n$, (2) each parameter vector and matrix of T is represented at p bits of precision, for some $p \in \mathbb{N}$ (3) Product functions involving only positions are translation invariant.*

Definition 7 (Transformer regularizer (Huang et al., 2025a, Definition 5))**.** *For $T \in \Theta_n^T$,*

$$\Re^T(T) := \mathcal{B}(T) + \max_{\ell,h} \text{rank}(V_{\ell,h}) + \max_{\sigma \in \Sigma} \|E_\sigma\|_2 + \sum_{\ell=1}^{L} \sum_{h=1}^{H} \sum_{j=1}^{n} |p_1^\top K_{\ell,h}^\top Q_{\ell,h} p_j|^2. \tag{8}$$

The last term discouraged distant positional interactions, thus in some sense inducing locality. Finally the inference procedure (very similar to Definition 4) was as follows.

Definition 8 (Transformer inference procedure (Huang et al., 2025a, Definition 6)). *Given a $f \in \mathcal{F}(\Sigma)$, the inference procedure generates a sequence of Transformers $T_1 \in \Theta_1^T, T_2 \in \Theta_2^T, \dots$ as follows. Define $\mathcal{U}_n^{\mathrm{T}}(f) := \{T \in \Theta_n^T : T \text{ matches } f \text{ on every input of length } \leq \lfloor n/2 \rfloor\}$. Then $T_n \in \mathcal{U}_n^{\mathrm{T}}(f)$ is chosesn such that*

$$\Re^T(T_n) \leq \frac{1}{n} + \inf_{T \in \mathcal{U}_n^{\mathrm{T}}(f)} \Re^T(T).$$

A.1.3 Growing Alphabet

The corresponding learning model from Sarrof et al. (2026) is as follows, we again paraphrase and define based on notation used in this paper. To model a growing vocabulary, the transformer operated on a split alphabet $\Omega = \Sigma \cup \mathcal{C}$, where $\mathcal{C} = \{c_1, c_2 \dots\}$ contains ordered characters that go upto infinity, similar to how the number of positions go to infinity earlier.

Definition 9 (Transformer hypothesis class (Sarrof et al., 2026, Definition C.7)). *For each $n = 1, 2, 3 \dots$, the hypothesis class $\Theta_n^{T,*}$ is the set of Transformers T, with $(1) N(T) = n$, (2) each parameter vector and matrix of T is represented at p bits of precision, for some $p \in \mathbb{N}$ (3) Product functions involving either only positions or only characters sourced from $\mathcal{C}$ are translation invariant.*

It should be noted that the original Sarrof et al. (2026, Definition C.7) allowed Transformers with the activation function $f(x) = 1/x$, which as we had remarked earlier, we forbid to ensure homogeneity (implications in Section A.1.4).

Due to the growing alphabet, the regularizer penalizes the product functions involving not just far away positions, but also far away characters (in terms of their order in $\mathcal{C}$). As the interactions between far away characters could happen not just through attention layers, but also through a path from a character starting from the embedding and ending up with a different character's unembedding, the product functions penalized include penalties of such terms. Thus, we denote these terms as follows.

1. $\alpha_{l,h}^{attn}(p1, p_j)$ is the sum of all product functions (all intermediate paths that connect the positional vectors to these nodes) between positions p_i, p_j through the attention head h in layer l.
2. $\alpha_{l,h}^{attn}(\omega_1, \omega_j)$ is the sum of all product functions (all intermediate paths that connect the character embeddings to these nodes) between characters ω_1, ω_k, where $\omega_1, \omega_k \in \Omega$.
3. $\alpha_{l,h}^{out}(U_{\omega_1}, E_{\omega_j}$ is the sum of all product functions (all intermediate paths that connects one character embedding to the unembedding of another) between any two characters $\omega_1, \omega_j \in \Omega$.

Therefore, with this the formal definition of the regularizer is as follows.

Definition 10 (Transformer regularizer (Sarrof et al., 2026, Definition C.8)). *For $T \in \Theta_n^{T,*}$,*

$$\Re^{T,*}(T) := \mathcal{B}(T) + \max_{\omega \in \Omega} ||E_\omega||_2 + \sum_{k=1}^{|\Omega_n|} |\alpha^{\mathrm{out}}(U_{\omega_1}, E_{\omega_k})|^2 \tag{9}$$

$$+ \sum_{\ell=1}^{L} \sum_{h=1}^{H} \left(\sum_{j=1}^{n} |\alpha_{\ell,h}^{\mathrm{attn}}(p_1, p_j)|^2) + |\alpha_{\ell,h}^{\mathrm{attn}}((\sum_{j=1}^{|\Omega_n|} U_{\omega_1}, E_{\omega_j})|^2 \right) \tag{10}$$

The inference procedure (very similar to Definition 5) is then as follows.

Definition 11 (Inference Procedure (Sarrof et al., 2026, Definition C.9)). *Given $f : (\Sigma \cup \mathcal{C})^* \to \{0, 1\}$, let $U_n^{f,*}$ be the subset of Transformers models $\Pi \in \Theta^{T,*}$ that matches f on the restricted domain of inputs, with (i) length of input x, $|x| \leq n/2$ (ii) the tokens $c_k \in \mathcal{C}$ appearing in x span a range of $n/2$, i.e. if $K_x = \{k : c_k \in x\}$ is the set of indices of those character tokens, then $\max K_x - \min K_x \leq n/2$. The inference procedure then generates a sequence $T_1 \in \Theta_1^{T,*}, T_2 \in \Theta^{T,*}, \dots$ by choosing $T_n \in U_n^{f,*}$ such that $\Re^R(T_n) \leq \frac{1}{n} + \inf_{T_n \in U_n^{f,*}} \Re^R(T_n)$.*

A.1.4 IMPLICATIONS OF REMOVING THE RECIPROCAL ACTIVATION FUNCTION

Sarrof et al. (2026) had defined a more general version of the match predicate in their C*-RASP[local] formulation as follows. Formally,

$$\chi(i,j) := \bigwedge_{k=1}^{K} (\boldsymbol{c}_{j-\delta_k} = \boldsymbol{c}_{i-\gamma_k} + \tau_k)$$

The use of the activation function $f(x) = 1/x$ in Sarrof et al. (2026) was *solely* used to prove that all of these K matches could be done at the same time. Thus the simple restriction of using the match predicate after setting $K = 1$ helps us get rid of this activation. Thus with our restriction, the match predictate becomes

$$\chi(i,j) := (\boldsymbol{c}_{j-\delta} = \boldsymbol{c}_{i-\gamma} + \tau)$$

Additionally, we write the match operation as $C(i) := \#\left[j \leq i, \chi(i,j)\right]\ P(j)$, instead of the $C(i) := \#\left[j \leq i, \chi(i,j)\right]$ defined in Sarrof et al. (2026) to mean $C(i) := \#\left[j \leq i, \chi(i,j) \wedge P(j)\right]$ just as syntactic sugar.

We note the following things about the existing results of Sarrof et al. (2026); Kraus et al. (2026), the papers which have so far used this formalism.

Remark 1. *1. Since our changes are only a subset of the original formalism, length generalization guarantees of Sarrof et al. (2026) hold for us.*

2. Kraus et al. (2026) had shown that C*-RASP[*local*] + CoT can compute any semi-computable function that can be computed a Turing machine. In their results as well, they did not use K matches together, hence their Turing completeness results still apply without considering the use of such activations.*

3. The result that Transformers can verify certain planning formalisms does not go through without the use of the full match predicate as it requires matching K matches together in the past.

Overall despite the removal of this activation, the C*-RASP[local] formalism stays quite powerful and captures the ability of Transformers to perform general induction heads and also the improvement in its performance upon having access to index hint like tokens (Zhou et al., 2024; Kraus et al., 2026). We therefore think that considering this restriction for the sake of homogenizing the frameworks across 3 different kinds of architectures is justified, and hope that in future work, the full power of C*-RASP[local] can be recovered without having to rely on $f(x) = 1/x$ as an activation function.

A.2 DPLR LINEAR RNNS

Here we will show how the formalizations of Gated DeltaNet (Yang et al., 2025b; Grazzi et al., 2025), RWKV-7 (Peng et al., 2025), Kimi Delta Attention (Kimi Team, 2025)are all instantiations of abstraction used in Equation 3, taken from Merrill et al. (2026a). For a common convention, we will use column vectors and a state $S_t \in \mathbb{R}^{d_s \times d_v}$, and the output of the head would be $y_t = S_t^\top q_t$. The State is initialized to be $S_0 = 0$ in each of these variants.

Gated DeltaNet. For each head, Gated DeltaNet (Yang et al., 2025b) produces a query and key $q_t, k_t \in \mathbb{R}^{d_s}$, a value $v_t \in \mathbb{R}^{d_v}$, and scalar gates $\alpha_t, \beta_t \in (0,1)$, (changed to $\beta_t \in (0,2)$ in Grazzi et al. (2025) to increase expressivity). The recurrence is then described as

$$S_t = \alpha_t\big(I - \beta_t k_t k_t^\top\big)S_{t-1} + \beta_t k_t v_t^\top, \qquad y_t = S_t^\top q_t. \tag{11}$$

Here, q_t, k_t, and v_t are obtained by a learned linear projection, a fixed-width short convolution, and SiLU; the query and key are then ℓ_2-normalized. The two gates are obtained from learned projections. By convention, Note that a zero vector remains zero even after normalization. Thus, Equation 11 is DPLR with rank one in the following way:

$$D_t = \alpha_t I, \qquad U_t = \alpha_t \beta_t k_t, \qquad V_t = k_t, \qquad B_t = \beta_t k_t v_t^\top, \qquad q(x_t) = q_t. \tag{12}$$

RWKV-7. RWKV-7 first forms a precursor k_t and then two versions of this key. In the notation of Peng et al. (2025),

$$\kappa_t = k_t \odot \xi, \qquad \widehat{\kappa}_t = \frac{\kappa_t}{\|\kappa_t\|_2}, \qquad \widetilde{k}_t = k_t \odot \mathrm{lerp}(\mathbf{1}, a_t, \gamma), \tag{13}$$

where $\widehat{\kappa}_t$ is the removal key, $\widetilde{k}_t$ is the replacement key, $a_t \in (0,1)^{d_s}$ is the in-context learning rate, lerp is a linear interpolation and ξ, γ are learned vectors. We use γ for the learned replacement-rate vector called α in Peng et al. (2025), to avoid confusing it with the decay used in the Gated DeltaNet architecture specification. At $\kappa_t = 0$, $\widehat{\kappa}_t = 0$. $w_t \in (0,1)^{d_s}$ denotes the decay, $r_t \in \mathbb{R}^{d_s}$ the query (called receptance in Peng et al. (2025), transforms the state to the output), and $v_t \in \mathbb{R}^{d_v}$ the value. The key, query, and value precursors are learned projections, and the recurrence is computed as follows,

$$S_t = \left(\mathrm{diag}(w_t) - (a_t \odot \widehat{\kappa}_t)\widehat{\kappa}_t^\top\right) S_{t-1} + \widetilde{k}_t v_t^\top, \qquad y_t = S_t^\top r_t. \tag{14}$$

RWKV-7 can be realized as DPLR with rank one as follows:

$$D_t = \mathrm{diag}(w_t), \qquad U_t = a_t \odot \widehat{\kappa}_t, \qquad V_t = \widehat{\kappa}_t, \qquad B_t = \widetilde{k}_t v_t^\top, \qquad q(x_t) = r_t. \tag{15}$$

Kimi Delta Attention. The recurrent layer introduced in Kimi Linear is called Kimi Delta Attention (KDA) (Kimi Team, 2025). It replaces Gated DeltaNet's scalar decay by a coordinatewise decay $\alpha_t \in (0,1)^{d_s}$:

$$S_t = \left(I - \beta_t k_t k_t^\top\right) \mathrm{diag}(\alpha_t) S_{t-1} + \beta_t k_t v_t^\top, \qquad y_t = S_t^\top q_t. \tag{16}$$

Here β_t is a scalar and q_t, k_t, v_t come from learned query, key, and value projections. These projections are followed by fixed-width depthwise short convolutions and SiLU, with zero-safe ℓ_2 normalization of the query and key; the other gates are obtained from learned projections. eThe recurrence in Equation 16 can be rewritten as

$$S_t = (\mathrm{diag}(\alpha_t) - (\beta_t k_t)(\alpha_t \odot k_t)^\top) S_{t-1} + \beta_t k_t v_t^\top, \qquad y_t = S_t^\top q_t. \tag{17}$$

This view makes it easy to see, that it is a DPLR with rank one where,

$$D_t = \mathrm{diag}(\alpha_t), \qquad U_t = \beta_t k_t, \qquad V_t = \alpha_t \odot k_t, \qquad B_t = \beta_t k_t v_t^\top, \qquad q(x_t) = q_t. \tag{18}$$

Support in LRNNs: We will now setup some definitions and notational choices that will come in handy in our proofs in Section B. We start by defining the support of a vector/ matrix as the set of its non zero entries:

$$\mathrm{supp}(x) := \{i : x_i \neq 0\}, \qquad \mathrm{supp}(X) := \{(i,j) : X_{ij} \neq 0\}$$

Thus $|\mathrm{supp}(x)|$ and $|\mathrm{supp}(X)|$ are the numbers of nonzero entries in x and X, respectively. We also define similar notions for counting number of rows/columns which are zero.

$$\mathrm{supp}_{\mathrm{row}}(X) := \{i : \|X_{i,:}\|_2 \neq 0\}, \qquad \mathrm{supp}_{\mathrm{col}}(X) := \{j : \|X_{:,j}\|_2 \neq 0\}$$

We can also extend the support notation to an entire model Π to say

$$|\mathrm{supp}(\Pi)| := \sum_{W \in \mathfrak{O}(\Pi)} |\mathrm{supp}(W)|$$

Here $\mathfrak{O}(\Pi)$ contains all the learned matrices, vectors, biases and convolution kernels of the model Π. We also count the learned parameters of the input dependent quantities in the DPLR LRNN head (eg: $D(z), U(z), V(z), A(z), q(z)$). For instance if $k(z) = \mathrm{SiLU}(W_k \varsigma(z) + b_k)$, then b_k and W_k would be counted in $\mathfrak{O}(\Pi)$, but not $\varsigma(z)$.

Let a_t be any of the vectors used by the LRNN head, such as a query, key or a value vector. We will learn projection which produces the precursor $\bar{a}_t = W_a \xi_{a,t} + b_a$. The output of this might be passes through a convolution as well, we can denote that by c_a, if its not passed through one, then $c_a = 0$. We can then define

Remark 2. *Let*

$$\mathcal{S}_a := \operatorname{supp}_{\text{row}}(W_a) \cup \operatorname{supp}(b_a) \cup \operatorname{supp}(c_a)$$

Therefore

$$|\mathcal{S}_a| \leq |\operatorname{supp}(W_a)| + |\operatorname{supp}(b_a)| + |\operatorname{supp}(c_a)|$$

Thus, every entry that is not in this set $\mathcal{S}_a$ remains zero in the vector generated from this projection irrespective of the input and the position.

Let $x_t(w)$ denote the input to a LRNN head at position t on input w, then the set of all inputs to a LRNN head is,

$$Z := \{x_t(w) : n \geq 1,\ w \in \Sigma^n,\ 1 \leq t \leq n\}. \tag{19}$$

Finally,

$$\mathbb{I} := \bigcup_{z \in Z} (\operatorname{supp}_{\text{row}}(U(z)) \cup \operatorname{supp}_{\text{row}}(V(z)) \cup \operatorname{supp}_{\text{row}}(B(z)) \cup \operatorname{supp}(q(z))), \tag{20}$$

$$\mathbb{J} := \bigcup_{z \in Z} \operatorname{supp}_{\text{col}}(B(z)). \tag{21}$$

Thus $\mathbb{I}$ contains all the rows of the state that are ever used (can be non zero) by a head and the set $\mathbb{J}$ contains every column of the state that is ever used by the components of the head.

Example: As this is a lot of notation, we give an example of how this would look like for a Gated DeltaNet head (Equation 12). We have,

$$\bar{k}_t = W_k x_t + b_k, \qquad \bar{q}_t = W_q x_t + b_q, \qquad \bar{v}_t = W_v x_t + b_v$$

The Gated DeltaNet applies SiLU, convolutions, and optionally for some also l_2 normalizes the vector, but all of these operations preserve zero coordinates, so

$$\operatorname{supp}(k_t) \subseteq \mathcal{S}_k, \qquad \operatorname{supp}(q_t) \subseteq \mathcal{S}_q, \qquad \operatorname{supp}(v_t) \subseteq \mathcal{S}_v$$

Thus,

$$\begin{aligned}
U_t = \alpha_t \beta_t k_t \quad &\Longrightarrow \quad \operatorname{supp}_{\text{row}}(U_t) \subseteq \mathcal{S}_k, \\
V_t = k_t \quad &\Longrightarrow \quad \operatorname{supp}_{\text{row}}(V_t) \subseteq \mathcal{S}_k, \\
B_t = \beta_t k_t v_t^\top \quad &\Longrightarrow \quad \begin{cases} \operatorname{supp}_{\text{row}}(B_t) \subseteq \mathcal{S}_k, \\ \operatorname{supp}_{\text{col}}(B_t) \subseteq \mathcal{S}_v, \end{cases} \\
q(x_t) = q_t \quad &\Longrightarrow \quad \operatorname{supp}(q(x_t)) \subseteq \mathcal{S}_q.
\end{aligned} \tag{22}$$

Consequently the row and column supports of the state of the head are

$$\mathbb{I} \subseteq \mathcal{S}_k \cup \mathcal{S}_q, \qquad \mathbb{J} \subseteq \mathcal{S}_v. \tag{23}$$

Before we start proving Theorem 1, we will first establish some facts about DPLR LRNNs. The first of which is that even on inputs of infinite length, the number of non zero entries in the state of DPLR Linear RNN head will be finite (even though the values at those coordinates might blow up, only capped by the maximum precision).

Lemma 1. *(Support of DPLR heads) For any of the DPLR variants of Gated DeltaNet, RWKV-7 and Kimi Delta Attention, there exists a constant c_{arch}, such that*

$$|\mathbb{I}| + |\mathbb{J}| \leq c_{arch} |\operatorname{supp}(\Pi)|$$

This constant depends on the exact specifics of the gates, normalizations used but is finite.

Proof. Recall that for each of the heads considered here, every entry outside the sets $\mathcal{S}_a$ (Remark 2), remains zero irrespective of the input and/or position. Thus now for everything inside this set. A depthwise convolution does not help mix entries in this set to any other coordinate of the matrix. Similarly $\mathrm{SiLU}(x) = x\,\mathrm{sigmoid}(x)$, $\mathrm{SiLU}(0) = 0$. Thus $\mathrm{supp}(a_t) \subseteq \mathcal{S}_a$ uniformly over all inputs and positions.

For Gated DeltaNet and Kimi Delta Attention, U_t, V_t are scalar or coordinatewise multiples of k_t, and B_t is a scalar multiple of $k_t v_t^\top$. For RWKV-7, both $\tilde{k}_t, \hat{k}_t$ are supported on $\mathcal{S}_k$, if we write $q = r$ for RWKV-7 query, then

$$\mathbb{I} \subseteq \mathcal{S}_k \cup \mathcal{S}_q \quad \mathbb{J} \subseteq \mathcal{S}_v$$

.

Multiplying a key coordinate with a gate does not increase its support, and diagonal decays also do not create new state rows. Each coordinate in $\mathcal{S}_a$ requires a nonzero projection coefficient or bias, and these parameters are all counted in $\mathrm{supp}(\Pi)$, so

$$|\mathbb{I}| + |\mathbb{J}| \leq |\mathcal{S}_k| + |\mathcal{S}_q| + |\mathcal{S}_v| \leq c_{\text{arch}} |\mathrm{supp}(\Pi)|$$

where c_{arch} depends on the exact parameters used in the head. □

B Main Proofs

B.1 Length Generalization in DPLR LRNNs

B.1.1 Finite Alphabet

We first look at the subspace in which the activations of the DPLR LRNN heads lie. First, the state space of our LRNN head is $\in \mathbb{R}^{d_s \times d_v}$. The *reachable span* (subspace of states which can be achieved given our initial state of $S_0 = 0$, and any input) of our state $\mathcal{M}$ is therefore:

$$\mathcal{M} = \mathrm{span}\left\{A(z_k)\cdots A(z_1)B(z_0) : k \geq 0,\ z_0, \ldots, z_k \in Z\right\} \tag{24}$$

Restricting to at most k transitions gives us the subspace $\mathcal{M}_k$.

$$\mathcal{M}_0 := \mathrm{span}\{B(z) : z \in Z\} \qquad \mathcal{M}_{k+1} := \mathcal{M}_k + \mathrm{span}\{A(z)S : z \in Z,\ S \in \mathcal{M}_k\}.$$

Lemma 2. *(Subspace of DPLR LRNN Heads) Let $\mathbb{R}^{\mathbb{I}\times\mathbb{J}}$ be defined as follows*

$$\mathbb{R}^{\mathbb{I}\times\mathbb{J}} := \{S \in \mathbb{R}^{d_s \times d_v} : S_{ij} = 0 \textit{ whenever } (i,j) \notin \mathbb{I}\times\mathbb{J}\}$$

The reachable subspace $\mathcal{M}$ is contained in $\mathbb{R}^{\mathbb{I}\times\mathbb{J}}$. Thus in particular, there exists a finite $\mathcal{K} \leq |\mathbb{I}||\mathbb{J}|$ such that $\mathcal{M}_k = \mathcal{M}_{k+1} = M_{k+2} = \cdots = \mathcal{M}$

Proof. Let $S \in \mathbb{R}^{\mathbb{I}\times\mathbb{J}}$. Since $D(z)$ is diagonal, $D(z)S$ cannot move a non zero value to a new row in the state. Similarly rows of $U(z)V(z)^\top S$ lie among the non zero rows of $U(z)$ and hence in $\mathbb{I}$, and this multiplication also leaves the zero columns outside $\mathbb{J}$ unchanged. Thus $A(z)S \in \mathbb{R}^{\mathbb{I}\times\mathbb{J}}$ for every $z \in Z$. Each $B(z)$ is also in the same space, and thus by induction, this space contains every product in Equation 24. Therefore $\mathcal{M} \subseteq \mathbb{R}^{\mathbb{I}\times\mathbb{J}}$ and therefore the effective dimension of $\mathcal{M}$ is bounded by $|\mathbb{I}||\mathbb{J}|$.

By definition,

$$\mathcal{M}_k = \mathrm{span}\left\{A(z_j)\cdots A(z_1)B(z_0) : 0 \leq j \leq k,\ z_0, \ldots, z_j \in Z\right\}.$$

Each $\mathcal{M}_0 \subseteq \mathcal{M}_1 \subseteq \cdots$. There will be some strict inclusions, and every strict inclusion will increase the dimension by at least one, and so the chain after $|\mathbb{I}||\mathbb{J}|$ steps stabilizes.

Finally, our $S_0 = 0$, therefore unrolling the recurrence gives

$$S_t = \sum_{i=1}^{t} A(x_t)A(x_{t-1})\cdots A(x_{i+1})B(x_i)$$

Every individual term being summed here is in $\mathcal{M}$, and so the state is also in $\mathcal{M}$. □

So far, we have not claimed that $|\mathbb{I}|$ or $|\mathbb{J}|$ are finite. But in fact we will now show that in our definition of the regularizer, bounding the regularizer bounds the number of non zero entries in the model, hence giving us a finitary characterization of Multilayer DPLR LRNNs.

Lemma 3. *For a given Multilayer LRNN model Π, and a regularizer $\Re$ as defined in Definition 3),*

$$\Re(\Pi) < \infty \to |\operatorname{supp}(\Pi)| < \infty$$

Proof. Recall, that the definition of our Regularizer is as follows

$$\Re(\Pi) := L + H + p(\Pi) + \sum_{W \in \mathfrak{D}(\Pi)} \|W\|_{\mathrm{F}} \tag{25}$$

All the terms here are non negative, hence when the regularizer is bounded by a finite value, i.e. $\Re(\Pi) < C$, then $L, H, p(\Pi) \leq C$ and also $\sum_{W \in \mathfrak{D}(\Pi)} \|W\|_{\mathrm{F}} < C$.

Since each non zero parameter has a minimum magnitude owing to finite precision, say η_C (as precision depends on the constant C), the total number of nonzero parameter entries are also bounded as well. Rearranging gives us $|\operatorname{supp}(\Pi)| \leq \frac{C^2}{\eta_C^2}$. [3] □

This Lemma 3 implies that a bounded regularizer bounds the number of non zero entries in each and every matrix/vector of the LRNN model. In particular this is also true for the LRNN head parameters. Therefore combining this with results of Lemma 3 and Lemma 2 implies that we have effectively only finitely many different algorithms that can be represented by a Multilayer RNN model (even though the total number of parameterizations are infinite).

With this, we are now ready to prove Theorem 1, we first restate it and then prove it.

Theorem 4. *(Idealized Length Generalization in the Limit for LRNNs) The inference procedure from Definition 4 applied to a target function $f \in \mathcal{F}(\Sigma)$ generates a sequence $\Pi_1, \Pi_2 \ldots$ with $\sup_{n=1,2,3} \Re(\Pi_n) < \infty$. If $f \in$ R-RASP, then there is some N_0 such that for all $m > N_0$, Π_m matches f on all inputs of any length $k \leq m$.*

Proof. As Lemma 3 establishes, under a bounded regularizer we are left with only finitely many algorithms, and thus amongst these finite algorithms, either some of them stop working at some N_0 or they match the behavior at all lengths. We now will show that indeed any R-RASP program can be encoded into the such finite LRNNs, by translating each R-RASP operation into a finite LRNN.

The MLP with the Heaviside activation in these models can be used to implement all the pointwise operations in R-RASP, using the same constructions as in Yang & Chiang (2024); Huang et al. (2025a); Yang et al. (2026b).

First we show how the finite-monoid operations and membership operations can be simulated.

Let M be any monoid, $\gamma\colon \{\top, \bot\}^k \to M$, and let $P_1, P_2, \ldots, P_k$ be R-RASP programs. Denote by $x_{i,c}$ the activation at position i in dimension c of a vector. Suppose inductively that there is a DPLR LRNN such that there are activations $x_{i,c_1}, \ldots, x_{i,c_k}$ such that $x_{i,c_l} = 1$ if $P_l(i) = \top$, and $x_{i,c_l} = 0$ if $P_l(i) = \bot$. We can add a layer to the DPLR LRNN which simulates the monoid operation as follows.

1. The hidden state will be $\mathbb{R}^{|M|}$. We will maintain the state so that one-hot vectors e_m for $m \in M$ indicate the prefix product.
2. $A(x_t)$ will only be a $x_{i,c_1}, \ldots x_{i,c_k}$. It can be defined as follows. Below we will use $\top = 1$ and $\bot = 0$ interchangeably. We will also use $m \in M$ and their index in S interchangeably.

$$A(x_{i,c_1}, \ldots x_{i,c_k}) = A_{\gamma(b_1,\ldots,b_k)}$$

$$A_m[m_1, m_2] = \begin{cases} 1 & m_1 m = m_2 \\ 0 & \text{otherwise} \end{cases}$$

[3] This is similar to how the number of active MLP neurons get bound in Huang et al. (2025a) owing to bounded frobenius norm of the MLP parameters.

In this way, $A_{m_1} e_{m_2} = e_{m_1 m_2}$. Then, given the state e_m, it is also easy for any $B \subseteq M$ to simulate a membership function that checks if $m \in M$ by choosing q such that $e_m q = e_1$ if $m \in B$ and $e_m q = e_0$ if $m \notin B$.

From the finite monoid operations we can simulate periodic operations by choosing a cyclic monoid $\mathbb{Z}_m$ for the period of the periodic operation.

Finally, we discuss on the counting operations. Consider a counting operation of the form $\#\,[j \leq i]\ \ P_l(j)$. In particular no local operations $\psi(i,j)$ are incorporated. Suppose $P_l(i) = \top$ iff $x_{i,d_l} = 1$ and $P_l(i) = \bot$ iff $x_{i,d_l} = 0$. By letting $S \in \mathbb{R}$ and $B_i = \begin{cases} 1 & x_{i,l} = 1 \\ 0 & \text{otherwise} \end{cases}$ We can have it so that S_i accumulates the sum of the counting operation.

In the case that we have a counting operation $\#\,[j \leq i, \psi(i,j)]\ \ P_l(j)$ with some local constraint $\psi(i,j)$ with window δ, we can actually simulate this with a finite monoid operation. This is because the value of the operation depends only on the sliding window of the δ most recently seen symbols. In particular we can write functions of such a fixed sliding window using monoid operations, with the underlying finite monoid we call here ∂_k. This monoid is formed as follows:

- Consider the set of all words $\{\top, \bot\}^*$ with concatenation as the operation
- Define an equivalence relation $w \equiv v$ iff the final k symbols of each word are identical (up to k, if the words have length shorter than k)
- Then ∂_k is the monoid of equivalence classes $\{\top, \bot\}^*$

In this way, a finite monoid operation using ∂_δ can simulate any operation over a finite sliding window of size δ. □

B.1.2 Growing Alphabet

We restate Theorem 2 and then prove it.

Theorem 5. *(Failure of LRNNs) There exists functions $f \in \mathcal{F}(\Sigma \cup \mathcal{C})$ where $f \in$ C*-RASP$_{atomic}$[local], and there is no sequence of LRNNs $\Pi_1, \Pi_2, \ldots$ with $\sup_{n \geq 1} \Re^R < \infty$ generated by the Inference procedure in Definition 5 such that after some $m > N_0$, $\overline{\Pi}_m$ matches f on inputs of any length $k \leq m$.*

Proof. Consider the following language $L \in \Omega^*$, where $L = \{w | w_{|w|} \in \mathcal{C} \wedge w_{|w|} \in w_{1,\ldots,|w|-1}\}$. Thus this language contains strings that end with a character sourced from our growing alphabet $\mathcal{C}$ and if this character was already present in our input string before this final position. Thus a very simple C*-RASP program for this is as follows.

$$C_{eq}(i) := \#\,[j \leq i, c_j = c_i] \qquad P_{final}(i) := \neg[C_{eq}(i) \leq 1]$$

Note that the input can have any character appearing in it till the last character, thus the LRNN model has no way of knowing which c_j to keep in memory, and it has to keep all the distinct tokens seen in an input in its hidden state to be able to solve the underlying task. Now in the extreme case, when the length of the input $n \to \infty$, then the allowed characters in our case $|C_n| \to \infty$, thus LRNNs have to maintain an increasingly growing number of alphabets in its hidden state. As these models, in the presence of a bounded regularizer have finitary characterization for algorithms that generalize and the only way for LRNNs to solve this task is to keep maintaining an increasing amount of memory, LRNNs fail to learn to solve this task length generalizably. □

B.2 Hybrid Models

We first give formal definitions for the learning framework of Hybrid models. To disambiguate between attention and recurrent heads we will refer to them with $\mathsf{A}(\Pi)$ and $\mathsf{R}(\Pi)$. These models use no positional embeddings: $x_t^{(0)} = E_{w_t}$, as LRNNs carry implicit positional information. All learned parameters have finite precision $p(\Pi)$. Following the transformer parameterization, the attention scores and exponentials are rounded to a fixed number of fractional bits. Attention is causal and uses

the $\log|w|$ scaling and softmax computation of Appendix A.1. Other intermediate computations use exact arithmetic.

Common parameters. Let $\mathfrak{O}_{\text{lrnn}}(\Pi)$ contain the learned parameters specific to recurrent heads: their input, query, key, value, gate, and output projections, biases, and convolution kernels. It does include shared embedding, unembedding and MLP parameters $E, U, C_\ell, D_\ell, b_\ell$, and also does not contain the values computed at inference time i.e. $A_t, B_t, D_t, U_t, V_t, q_t$. Define

$$\mathcal{B}_{\text{H}}(\Pi) := L + H + \rho(\Pi) + \max_{(\ell,h)\in\mathsf{A}(\Pi)} \|K_{\ell,h}^\top Q_{\ell,h}\|_{\text{op}} + \max_{(\ell,h)\in\mathsf{A}(\Pi)} \|V_{\ell,h}\|_{\text{op}} + \max_\ell \|C_\ell\|_{\text{F}} + \max_\ell \|D_\ell\|_{\text{F}} + \max_\ell \|b_\ell\|_2 + \|U\|_{\text{op}} + \sum_{W\in\mathfrak{O}_{\text{lrnn}}(\Pi)} \|W\|_{\text{F}}. \tag{26}$$

The inference procedure for the Hybrid models for both finite and growing alphabet are identical to the ones in Definition 4 and Definition 5, except that we use the relevant hypothesis class and regularizer for Hybrid models, which we define next.

B.2.1 Finite Alphabet

Definition 12 (Hybrid hypothesis class, finite alphabet)**.** *The hypothesis class Θ^H is the set of Hybrid models Π operating over a finite alphabet Σ, composed of layers with either softmax attention heads (as defined in Equation 5) or LRNN heads (as defined in Equation 2) in any arbitrary order with (1) each parameter vector and matrix of Π is represented at a finite p bits of precision.*

As positional embeddings are not used, the definition of the hypothesis class over a finite alphabet is considerably simpler and additionally the regularizer also then does not include the product function sum over positions to penalize distant positional interactions.

Definition 13 (Regularizer for Hybrid Models over Σ)**.** *For $\Pi \in \Theta^H$, define*

$$\mathfrak{R}^H(\Pi) := \mathcal{B}_{\text{H}}(\Pi) + \max_{\sigma\in\Sigma} ||E_\sigma||_2 + \max_{(\ell,h)\in\mathsf{A}(\Pi)} \text{rank}(V_{\ell,h})$$

B.2.2 Growing Alphabet

Definition 14 (Hybrid Models hypothesis class, growing alphabet)**.** *For each $n = 1, 2, 3\ldots$ the hypothesis class $\Theta_n^{H,*}$ is the set of Hybrid models Π operating over the alphabet $\Omega = \Sigma \cup \mathcal{C}_n$, where Σ is fixed and $\mathcal{C} = c_1, c_2, \ldots c_n$, each parameter vector and matrix of Π is represented at p bits of precision, for some $p \in \mathbb{N}$ (3) Product functions involving characters sourced from $\mathcal{C}$ are translation invariant.*

Definition 15 (Hybrid Models regularizer, growing alphabet)**.** *For $\Pi \in \Theta^{H,*}(\Omega_n)$*

$$\mathfrak{R}^{T,*}(T) := \mathcal{B}(T) + \max_{\omega\in\Omega} ||E_\omega||_2 + \sum_{k=1}^{|\Omega_n|} |\alpha^{\text{out}}(U_{\omega_1}, E_{\omega_k})|^2 + \sum_{\ell=1}^{L}\sum_{h=1}^{H}\sum_{j=1}^{|\Omega_n|} |\alpha_{\ell,h}^{\text{attn}}(U_{\omega_1}, E_{\omega_j})|^2$$

C Task Descriptions and RASP expressiveness

We now describe each task used in our experiments, and along with that in tables 2–5 we mention the membership (if it exists) of the language in the smallest RASP slice amongst C-RASP[periodic, local], R-RASP, C*-RASP$_{\text{atomic}}$[local], and H-RASP. As we are denoting the smallest RASP slice in which a language belongs, the expectations based on RASP membership are as follows. Being placed in C-RASP[periodic, local] would imply expected empirical length generalization for Transformers, LRNNs and Hybrids, being placed in R-RASP would imply expected generalization in LRNNs and Hybrids, being placed in C*-RASP$_{\text{atomic}}$[local] would imply generalization in Transformers and Hybrids and finally in H-RASP would imply just Hybrids generalizing well.

C.1 Regular and Counter Languages considered

C.1.1 Regular Languages

Regular language	Membership	Reason
Tomita 1	C-RASP[periodic, local]	Test for symbols other than 1.
Tomita 2	C-RASP[periodic, local]	Prefix counts enforce alternation.
Tomita 3	R-RASP	Regular; outside C-RASP[periodic, local].
Tomita 4	C-RASP[periodic, local]	Local tests exclude 000.
Tomita 5	R-RASP	Regular; parity excludes C-RASP[periodic, local].
Tomita 6	R-RASP	Regular; modulo-three counting excludes C-RASP[periodic, local].
Tomita 7	C-RASP[periodic, local]	Test the order of symbol runs.
D_2	C-RASP[periodic, local]	Prefix balance with depth at most 2.
D_3	C-RASP[periodic, local]	Prefix balance with depth at most 3.
D_4	C-RASP[periodic, local]	Prefix balance with depth at most 4.
D_{12}	C-RASP[periodic, local]	Prefix balance with depth at most 12.
$(aa)^*$	C-RASP[periodic, local]	Periodic length test, modulo 2.
$(aaaa)^*$	C-RASP[periodic, local]	Periodic length test, modulo 4.
$(abab)^*$	C-RASP[periodic, local]	Alternation and a periodic length test.
$aa^*bb^*cc^*dd^*ee^*$	C-RASP[periodic, local]	Test symbol occurrence and order.
$\{a, b\}^*d\{b, c\}^*$	C-RASP[periodic, local]	Test the unique d and surrounding symbols.
$\{0, 1, 2\}^*02^*$	R-RASP	Regular; the FlipFlop obstruction excludes C-RASP[periodic, local].

Table 2: RASP membership for the 17 regular languages. These memberships and exclusions are from Huang et al. (2025a, Table 2 and Appendix E.1.2); R-RASP membership follows from the fact that all languages are regular and that regular languages by definition only have finite alphabet.

Each of the languages in the regular language suite from Bhattamishra et al. (2020); Huang et al. (2025a) are described here as follows. Given a source string w, each position i is labeled with a finite-state annotation of the prefix $w_{1:i}$: either the set of legal next symbols (packed as one vocabulary token), prefix membership in the language or a state representation for the prefix. Next-symbol prediction on these languages is generally not deterministic, so the model is trained to predict these annotations rather than next word tokens. Labels are aligned with word prefixes changing the setup slightly compared to algorithmic tasks:

```
input    <bos>  1  0  1  0  1  0 <eos>
target     -   yn ny yn ny yn ny   -
```

All tokens except for '−' are padded and calculated in the accuracy score.

What the labels mean. Tokens represent binary inclusion or exclusion (`ny` or `01`) over the languages' alphabets for legal next symbol predictions. The same binary alphabets are used for prefix membership.

Tomita 1. 1^*. Legal next symbols are always 1 (`ny`).

```
input    <bos>  1  1  1  1  1 <eos>
target     -   ny ny ny ny ny   -
```

Tomita 2 $(10)^*$. Legal next symbols alternate: after 1 only 0 (`yn`), after 0 only 1 (`ny`).

```
input    <bos>  1  0  1  0  1  0 <eos>
target     -   yn ny yn ny yn ny   -
```

Tomita 3. *Strings with no odd-length run of* 1*s followed by an odd-length run of* 0*s.* After a single 1 there may not be a 0 (ny):

```
input    <bos>  1  1  1  0  0 <eos>
target     -   ny yy ny yn ny   -
```

Tomita 4. *Strings containing no substring* 000*.* After two consecutive 0s only 1 is legal (ny). A 1 resets the run:

```
input    <bos>  1  1  0  0  1 <eos>
target     -   yy yy yy ny yy   -
```

Tomita 5. *Strings with an even number of* 0*s and an even number of* 1*s.* Labels are prefix membership (y/n).

```
input    <bos> 0 0 1 1 <eos>
target     -   n y n y   -
```

Tomita 6. *Strings with* $|\#0 - \#1|$ *divisible by* 3*.* Labels are prefix membership.

```
input    <bos> 1 1 0 1 1 <eos>
target     -   n n n n y   -
```

Tomita 7. $0^*1^*0^*1^*$. Once the second run of 1s has started, 0 is illegal (ny on the last token):

```
input    <bos>  0  0  1  1  0  1 <eos>
target     -   yy yy yy yy yy ny   -
```

D_n**.** *Bounded-depth Dyck-1 over* $\{a, b\}$*, defined by* $D_0 = \varepsilon$ *and* $D_n = (a\, D_{n-1}\, b)^*$. The label at each position is the depth state of the prefix: 10 if the current depth is 0, 01 if it equals n, and 11 otherwise. The tasks D_2, D_3, D_4, D_{12} differ only in the depth bound n.

D_2 on *aabbab* reaches the cap $n = 2$ on the second a (01) and returns to depth 0 on each closing b that balances a full D_2 factor (10):

```
input    <bos>  a  a  b  b  a  b <eos>
target     -   11 01 11 10 11 10   -
```

D_3 on *aaabbb*:

```
input    <bos>  a  a  a  b  b  b <eos>
target     -   11 11 01 11 11 10   -
```

D_4 on *aaaabbbb*:

```
input    <bos>  a  a  a  a  b  b  b  b <eos>
target     -   11 11 11 01 11 11 11 10   -
```

D_{12} on *aaaabbbb*. Strings shorter than 24 tokens never reach the depth-cap and are only labeled with 11 and 10:

```
input    <bos>  a  b  a  b <eos>
target     -   11 10 11 10   -
```

$(aa)^*$**.** Each prefix is labeled 1 iff its length is even.

```
input    <bos> a a a a a a <eos>
target     -   0 1 0 1 0 1   -
```

$(aaaa)^*$. Each prefix is labeled 1 iff its length is divisible by 4.

```
input    <bos> a a a a a a a a <eos>
target     -   0 0 0 1 0 0 0 1 -
```

$(abab)^*$. Prefix membership: 1 only at multiples of 4.

```
input    <bos> a b a b a b a b <eos>
target     -   0 0 0 1 0 0 0 1   -
```

$aa^*bb^*cc^*dd^*ee^*$. The target at each position is the next letter of this sorted-run language, written in a disjoint alphabet $\{B, C, D, E, T\}$ (T is the terminator after e).

```
input    <bos> a a b b c c d d e e <eos>
target     -   B B C C D D E E T T   -
```

$\{a, b\}^*d\{b, c\}^*$. Each position is labeled with a 4-bit mask over (a, b, c, d): `1101` (legal $\{a, b, d\}$) before the unique d, and `0110` (legal $\{b, c\}$) from d onward.

```
input    <bos>    a    b    d    b    c    c <eos>
target     -   1101 1101 0110 0110 0110 0110   -
```

$\{0, 1, 2\}^*02^*$. The label is `c` (must continue: $0, 1, 2$ are all still possible) until a 0 has been seen and only 2s have followed it, after which `e` marks that the string may end.

```
input    <bos> 1 2 0 2 2 <eos>
target     -   c c c e e   -
```

C.1.2 Counter Languages

Counter language	Membership	Reason
Dyck-1	C-RASP[periodic, local]	Prefix balance of opens and closes.
Shuffle-2	C-RASP[periodic, local]	Two independent bracket balances.
Shuffle-4	C-RASP[periodic, local]	Four independent bracket balances.
Shuffle-6	C-RASP[periodic, local]	Six independent bracket balances.
Boolean-3	C-RASP[periodic, local]	Count unfilled arguments using token arities.
Boolean-5	C-RASP[periodic, local]	Count unfilled arguments using token arities.
$a^n b^n$	C-RASP[periodic, local]	Compare symbol counts and check block order.
$a^n b^n c^n$	C-RASP[periodic, local]	Compare symbol counts and check block order.
$a^n b^n c^n d^n$	C-RASP[periodic, local]	Compare symbol counts and check block order.

Table 3: All nine counter-language tasks lie in C-RASP[periodic, local].

Dyck-1 The task is to predict the next possible token in a string containing matched parentheses of one type. Label 1 indicates that the next generated symbol can be either (or), whereas label 0 indicates that it can be either (or $\langle\text{eos}\rangle$.

```
input    <bos> ( ( ( ) ) ) <eos>
target     -   1 1 1 1 1 0 -
```

Shuffle-2 The task is to predict the next possible token in a string that contains matched parentheses of two types, and the parentheses should be matched only within its type. The first digit in the label corresponds to parentheses of type (, and the second digit – to parentheses of type [. Label 1 indicates that the next generated symbol can be either a closing or an opening bracket of that type, label 0 indicates that it can be an opening bracket only. Label 00 means that the next symbol can be either an opening bracket of any type, or ⟨eos⟩.

```
input    <bos> [  (  )  ]  <eos>
target     -   01 11 01 00 -
```

Shuffle-4 Same as shuffle-2 (C.1.2), but the string contains matched parentheses of four types, and the parentheses should be matched only within its type. The label thus has four digits – one per type of parentheses – and 0000 label indicates that the next token can be either an opening bracket, or ⟨eos⟩.

```
input    <bos> {    [    ]    }    [    ]    <eos>
target     -   0010 0110 0010 0000 0100 0000 -
```

Shuffle-6 Same as shuffle-2 (C.1.2), but the string contains matched parentheses of six types, and the parentheses should be matched only within its type. The label thus has six digits – one per type of parentheses – and 000000 label indicates that the next token can be either an opening bracket, or ⟨eos⟩.

```
input    <bos> a      <      >      b      <eos>
target     -   000001 000101 000001 000000 -
```

Boolean-3 The task is to predict whether the input is a valid boolean expression written in prefix form with operands of arity up to 3. It can have the following operands: ∼ which takes 1 argument, + which takes 2 arguments, and > which takes 3 arguments, as well as 0 and 1 which take zero arguments. In prefix form 'apply operand + to arguments 0 and 1' can be written as '+01'. We write label 1 at a given position if the current substring is a valid boolean expression, and 0 otherwise.

```
input    <bos> ~ + 0 > 0 0 ~ 0 <eos>
target     -   0 0 0 0 0 0 0 1 -
```

Boolean-5 The task is to predict whether the input is a valid boolean expression written in prefix form with operands of arity up to 5. It can have the following operands: ∼ which takes 1 argument, + which takes 2 arguments, and > which takes 3 arguments, ˆ which takes 4 arguments, ∗ which takes 5 arguments, as well as 0 and 1 which take zero arguments. In prefix form 'apply operand + to arguments 0 and 1' can be written as '+01'. We write label 1 at a given position if the current substring is a valid boolean expression, and 0 otherwise.

```
input    <bos> * 0 0 1 0 0 <eos>
target     -   0 0 0 0 0 1 -
```

$a^n b^n$ The task is to predict the next possible token in a valid string in $a^n b^n$ language, i.e. a language with equal number of a's and b's. Label 'a/b' indicates that the next token can be either a or b, 'b' indicates that it should be b, and 'T' means the next token should be ⟨eos⟩.

```
input    <bos> a a   a   b   b b <eos>
target     -   - a/b a/b a/b b b T
```

$a^n b^n c^n$ Same as $a^n b^n$ (C.1.2), but instead of equal number of a's and b's the language contains strings with equal number of a's, b's and c's.

```
input    <bos> a   a   b b c c <eos>
target     -   a/b a/b b c c T -
```

$a^n b^n c^n d^n$ Same as $a^n b^n$ (C.1.2), but instead of equal number of a's and b's the language contains strings with equal number of a's, b's, c's and d's.

```
input    <bos> a   a   b b c c d d <eos>
target     -   a/b a/b b c c d d T -
```

C.2 Algorithmic Tasks

Algorithmic task	Membership	Reason / reference
Parity	R-RASP	Regular; outside C-RASP[periodic, local] (Huang et al., Lemma 41).
Addition (serialized)	None	Linear communication: $x + 0$ requires copying x; Theorem 3.
Binary Majority	C-RASP[periodic, local]	Compare the counts of 0 and 1.
Majority (fixed alphabet)	C-RASP[periodic, local]	Compare counts for each of finitely many symbols.
Binary Majority Interleave	C-RASP[periodic, local]	Periodic predicates separate the three streams.
Unique Copy (growing alphabet)	C*-RASP$_{\text{atomic}}$[local]	Equality matching and local shifts; (See Program C.2).
Repeat Copy	None	Linear communication for unrestricted copying; Theorem 3.
Set Sort	C-RASP[periodic, local]	Check presence of alphabets in some predefined order in the program, outputting alphabets if they appear in the input

Table 4: RASP membership for the algorithmic suite. Unless specified, reasons are taken from Huang et al. (2025a, Table 3 and Appendix E.2.2).

Algorithmic tasks are trained as causal language models on prompt-answer inputs

```
input     <bos> x <sep> y <eos>
target     -   -   y <eos> -
```

except Addition, which uses "=" in place of <sep>. The string x is the task input, and y is the answer token or sequence. All dashes '$-$' mark masked tokens, which are padded and not scored against, such that cross-entropy loss is only computed for y and <eos>.

Resolving ambiguity about Sort and Unique Copy. Huang et al. (2025a) had claimed that both Unique copy and Sort have programs in C-RASP[periodic, local], but had left the descriptions of these tasks implicit, in how they a language with an infinite alphabet might have programs in a logic operating over a finite alphabet. While for Unique copy, there is no way to define this language, such that longer lengths do not use a growing alphabet, and therefore we accordingly give C*-RASP[local] programs for it. However the Sorting task can be defined in terms of a finite alphabet as well, thus we interpret their definition of Sort, as being Set Sort, which we define next, as the C-RASP program given for Sorting in Huang et al. (2025a) would only work over a finite alphabet, and would otherwise be an infinite length program.

Set Sort Let x be a string over an alphabet, which is totally ordered (i.e. integer-like tokens) with repetitions allowed. Then performing Sort is to output the unique tokens in x in increasing order.

```
input    <bos> 9 36 9 42 20 42 <sep> 9 20 36  42 <eos>
target     -   -  -  - -  -  -   9  20 36 42 <eos> -
```

Parity. On a vocabulary $\{0, 1\}$, the model must predict whether a sequence contains an even or odd number of 1s. The answer vocabulary is $\{\texttt{e}, \texttt{o}\}$, even or odd. For a word including three 1s the model must predict:

```
input    <bos> 0 0 0 1 1 0 1 <sep>  o <eos>
target     -   - - - - - - -   o  <eos> -
```

Addition. Let x be the input sequence $x = a_1, ..., a_n, +, b_1, ..., b_m, =$ encoding two binary numbers. Let A and B be the integers represented by the binary strings $a_1, ..., a_n$ and $b_1, ..., b_n$ respectively. Performing Addition is then to output the binary string representation of $A + B$. Note that the $=$ token is used here for seperation.

In the below example, the model must predict $1110_2 + 101_2 = 14 + 5 = 19 = 10011_2$.

```
input    <bos> 1 1 1 0 + 1 0 1 = 1 0 0 1   1 <eos>
target     -   - - - - - - - - 1 0 0 1 1 <eos> -
```

Binary Majority. Given a word x over a binary alphabet $\{0, 1\}$, the model must output the more frequent token in x. Ties are rejected at sampling.

For four 1s and three 0s the answer is 1:

```
input    <bos> 1 1 0 1 0 0 1 <sep>  1 <eos>
target     -   - - - - - - -   1  <eos> -
```

Majority. Given a word x over alphabet V, the model must output the most frequent token in x. Ties are rejected at sampling.

Training uses V to be the 26 lowercase letters. The letter occurring the most often below is then `i`:

```
input    <bos> g i l v c i k t <sep>  i <eos>
target     -   - - - - - - - -   i  <eos> -
```

Binary Majority Interleave. Every word x consists of p subsequences $s^{(1)}, \ldots, s^{(p)}$ each of which are over the same binary alphabet. All subsequence are interleaved in the following way: $[s_1^{(1)}, s_1^{(2)}, ..., s_1^{(p)}, s_2^{(1)}, ..., s_L^{(p)}]$, where L is the length of each subsequence. Performing Binary Majority Interleave is outputting a sequence of length p of the most frequent element for each subsequence (Binary Majority per subsequence).

Training uses three subsequences ($p = 3$). Below the streams are $(0, 1, 0)$, $(1, 0, 0)$, $(1, 1, 1)$, with majorities $0, 0, 1$:

```
input    <bos> 0 1 1 1 0 1 0 0 1 <sep> 0 0   1 <eos>
target     -   - - - - - - - - -   0   0 1 <eos> -
```

Unique Copy. Let x be a string of unique tokens, then the model must output x.

For training, tokens are distinct integers $\{0, \ldots, L_{\max} - 1\}$, where $L_{\max}$ is the largest evaluation size.

```
input    <bos> 21 46 28 9 27 <sep> 21 46 28 9   27 <eos>
target     -    -  -  - - -   -   21  46 28  9 27 <eos> -
```

In this case, we can formalize the problem as a decision problem to show the program (similar to Sarrof et al. (2026)). Essentially, it suffices to show that each bigram that occurs after the separator token occurs exactly once before the separatior

Unique copy

$$C_{<sep}(i) := \#\,[j \le i]\ Q_{\texttt{sep}}(j) \tag{1}$$
$$P_{<sep}(i) := C_{<sep}(i) \ge 1 \tag{2}$$
$$C_{bigram}(i) := \#\,[j \le i, c_{j-1} = c_{i-1}, c_j = c_i]\ P_{<sep}(j) \tag{3}$$
$$U_{bigram}(i) := C_{bigram}(i) = 1 \tag{4}$$
$$C_{violate}(i) := \#\,[j \le i]\ \neg U_{bigram}(j) \tag{5}$$
$$P(i) := C_{violate}(i) = 0 \tag{6}$$

Unique Suffix Copy. Let x be a string of unique symbol tokens, with some index tokens interspersed. Then the model must find the rightmost index token j, and output x from position j onwards.

For training, tokens are distinct from some set V of size at least L. We have a vocabulary of finitely many markers $\mathcal{M} = \{m_1, \ldots, m_k\}$.

```
input    <bos> a 4 b c 3 d e f <sep> b c 3 d e f     <eos>
target     -   - - - - - - - - b     c 3 d e f <eos> -
```

Consider the monoid M over $\{e\} \cup \{m_1, \ldots, m_k\}$, where e acts as the identity and $m_i \cdot m_j = m_j$. Let $\gamma\colon (V \cup \mathcal{M}) \to M$ given by $\gamma(v) = 1$ for $v \in V$ and $\gamma(m_i) = m_i$. This monoid operation can be used to detect the right-most marker m_r. Using m_r, we can write operations $P(i)$ that are true iff $i \le m_r$. This allows us to mask out all positions with index $\le m_r$. Finally, a modified program for Unique Copy can be used to copy the remaining unmasked symbols.

We note that this language is not definable in R-RASP, because no R-RASP program can refer to symbols of the growing alphabet. We also note that no C-RASP[periodic, local] program exists either. First, let $V = \{a, b, c\}$ and $\mathcal{M} = \{1, 2\}$ and consider any C-RASP[periodic, local] program correctly predicting the first symbol after sep. In this case, if we fix a prefix ab the task essentially requires us to be able to recognize the language $ab(V \cup \mathcal{M})^* 2 V^* \texttt{sep}$ (in order to determine whether the next symbol is a or b). The inexpressivity of this language in C-RASP[periodic, local] was shown by Huang et al. (2025a, Lemma 36).

Repeat Copy. Let $x \in V^L = x_1, ..., x_L$ where repetition amongst x_i is allowed, then RepeatCopy is the task to output x. Let x be an arbitrary string, where repititions of tokens are allowed. The correct output is then x.

Inputs are structured as for UniqueCopy, except that the alphabet is $\{a, b\}$ so repetitions occur. The model must copy every token, including repeats:

```
input    <bos> b a a a b a <sep> b a a a b   a <eos>
target     -   - - - - - -   b   a a a b a <eos> -
```

C.3 Additional Tasks considered

Task	Membership	Reason
S_5 State Tracking	R-RASP	Regular language
Multiplication (serialized)	None	Linear communication: $x \times 1$ requires copying x; Theorem 3.
Digit-by-digit addition (Mohri, 2026)	R-RASP	is regular, DFA defined in paragraph C.3
Selective Copy (Cooper et al., 2026)	R-RASP	See Paragraph C.3
Unique Suffix Copy	H-RASP	See Paragraph C.2

Table 5: RASP memberships for the additional tasks considered.

Selective Copy. The task is defined over a marker vocabulary $M = \{\#1, ..., \#k\}$, each of which can be seen to represent an integer number, as well as a filler vocabulary F (i.e. $\{f_0, ..., f_n\}$). Given a sequence of markers and fillers, the model must output the m-th to last token in the sequence, where m is the integer value of the last occuring marker in the sequence.

Below the last marker is $\#6$, so the answer is the sixth-from-last token, here $\#1$.

```
input    <bos> m0 #2 #1 m0 m1 m2 m0 #6 <sep> #1 <eos>
target     -    -  -  -  -  -  -  -  -  #1 <eos>  -
```

A solution is definable in R-RASP. First, similar to the solution for unique suffix copy (C.2), we can write a monoid operation that find the right-most marker token. This allows us to define an operation $P_m(i)$ which is true at position i iff the rightmost marker to the left of i is $\#m$. Then, for each $\sigma \in \{\#1, \ldots, \#k, f_0, \ldots, f_n\}$ we can write a counting operation $C_\sigma(i) := \#\,[j \leq i, j = i - m]\ Q_\sigma(j)$ that checks if the m-th to last token is σ, using $C_\sigma(i) \geq 1$. Running each $C_\sigma(i)$ at the `sep` position, we can thus predict the next token to solve the selective copy task.

State Tracking We use the swaps-only variant of the S5 state tracking task from Grazzi et al. (2025). S_5 refers to the group of permutations of 5 elements. Each input token g_t is sampled uniformly from the identity e and the ten transpositions (swaps of two elements). Starting from $p_0 = e$, the model has to predict the intermediate permutation $p_t = p_{t-1} \circ g_t$ at every position, where $(p \circ g)(i) = p(g(i))$, the answer can be any of the $5! = 120$ tokens. In the example below, we show permutation tokens in cycle notation: $(1\,2)$ swaps elements 1 and 2, and e leaves the state unchanged.

```
input    <bos> (1 2) (2 3)    e       (1 2) <eos>
target     -   (1 2) (1 2 3) (1 2 3) (1 3)   -
```

Multiplication. Let x be the input sequence $x = a_1, ..., a_n, *, b_1, ..., b_m, =$ encoding two binary numbers. Let A and B be the integers represented by the binary strings $a_1, ..., a_n$ and $b_1, ..., b_m$ respectively. Performing Multiplication is then to output the binary string representation of $A * B$. Note that the $=$ token is used here for separation. The product of an n-bit number and an m-bit number has at most $n + m$ bits.

In the below example, the model must predict $1110_2 * 101_2 = 14 * 5 = 70 = 1000110_2$.

```
input    <bos> 1 1 1 0 * 1 0 1 = 1 0 0 0 1 1   0 <eos>
target     -   - - - - - - - - 1 0 0 0 1 1 0 <eos> -
```

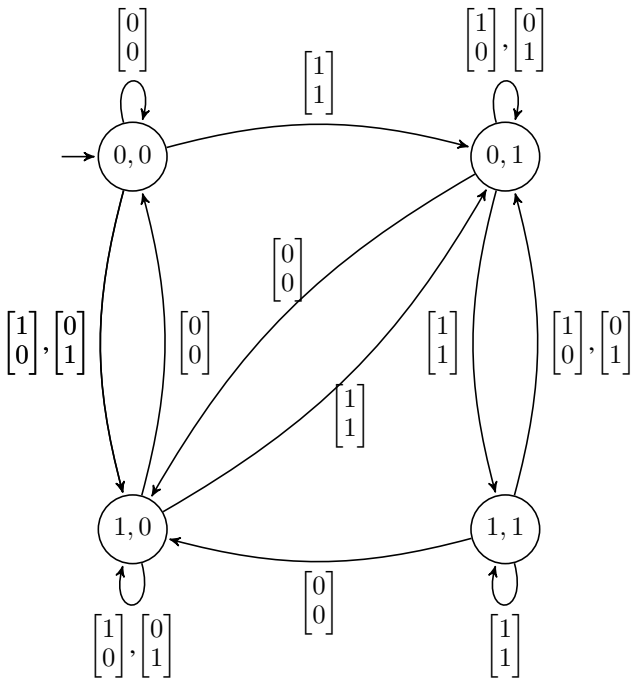
Digit by Digit Addition Given two numbers, a and b, in binary encoding, the task is to output c, their sum in binary encoding, c = a + b. Suppose a consists of digits $\overline{a_n, \ldots, a_1}$, $b = \overline{b_m, \ldots, b_1}$, and $c = \overline{c_{\max(n,m)+1}, \ldots, c_1}$, where a_1, b_1 and c_1 are the lowest bits in respective numbers. Without loss of generality, assume $n > m$, and then we pad b with leading zeros: $b = \overline{0 \ldots 0 b_m, \ldots, b_1} = \overline{b_n, \ldots, b_1}$. We encode this summation to input it in the model like this: $[a_1 b_1], [a_2 b_2], \ldots, [a_n b_n], [00]$, where brackets indicate one token. The task is to predict the digits of the sum c. Specifically, from token $[a_1 b_1]$ the model has to predict c_1, from token $[a_n b_n]$ – c_n, and from the trailing $[00]$ it has to predict c_{n+1}.

In the example below $a = 0, b = 1, c = 1$.

```
input    <bos> 01 00 <eos>
target     -   1  0  -

input    <bos> 0 1 0 0 <eos>
                 1 1 1 0
target     -   1 0 0 1 0
```

The next token prediction can be simulated by a finite state machine, which we display below. Each state indicates the output bit and the carry bit. Knowing the DFA, it is easy to write a finite monoid operation in H-RASP in order to solve the task.



D Additional Experimental Details

D.1 Details on Training Setup

This appendix explains the details behind Section 4. Every model family (Transformer, SSM (LRNN), and Hybrid) is run on every task. For each model-task pair we sweep layers (or motif repetitions), heads, embedding dimension, learning rate and dropout (Table 6), and additionally the positional-encoding variant (RoPE/NoPE on the OLMo-core stack, APE/NoPE on the GPT-2 stack).

Model	Layers	Width	Heads	Dropout	LR	PE
Hybrid	1, 2, 4	16, 64	1, 2, 4	0, 0.1	10^{-3}, 10^{-4}	NoPE
Transformer	1, 2, 4, 8*	16, 32, 64, 256	1, 2, 4	0, 0.1	10^{-3}, 10^{-4}	NoPE, RoPE, APE
SSM	1, 2, 4, 8*	8, 16, 32, 64	–	0, 0.1	10^{-3}, 10^{-4}	none

Table 6: Sweep for each model family. Each row is the cross product of its entries. Hybrid layers are repetitions of a recurrence-attention stack. The default budget is 30k steps. Configurations with more than four Transformer or SSM layers, or more than two Hybrid stacks, use 60k steps. Tasks which were empirically found to be hard to train were trained for up to 120k steps. * Eight layers are only used for negative examples.

Data generation and evaluation bins. Training data for all tasks are sampled on the fly: for every training example we first draw a length uniformly from the training window and then sample a word of that length. For algorithmic tasks evaluation bins are drawn once up front and then reused during training for every evaluation step, so that all accuracies for a single sweep are measured on identical words. Evaluation bins are within the intervals of $[L_{min}, 50]$, $[51, 100]$, $[101, 150]$, each consisting of 2000 words, where L_{min} is each language's smallest possible word length (i.e. 4 for Addition and Multiplication). An exception is $a^n b^n$, $a^n b^n c^n$, and $a^n b^n c^n d^n$ languages – they were trained on lengths up to 200 and evaluated on lengths up to 600, since for them there are too few available training examples of length up to 50. Regular languages instead use per-language training-windows and evaluation sizes indicated in Tables 12, 11, and 13.

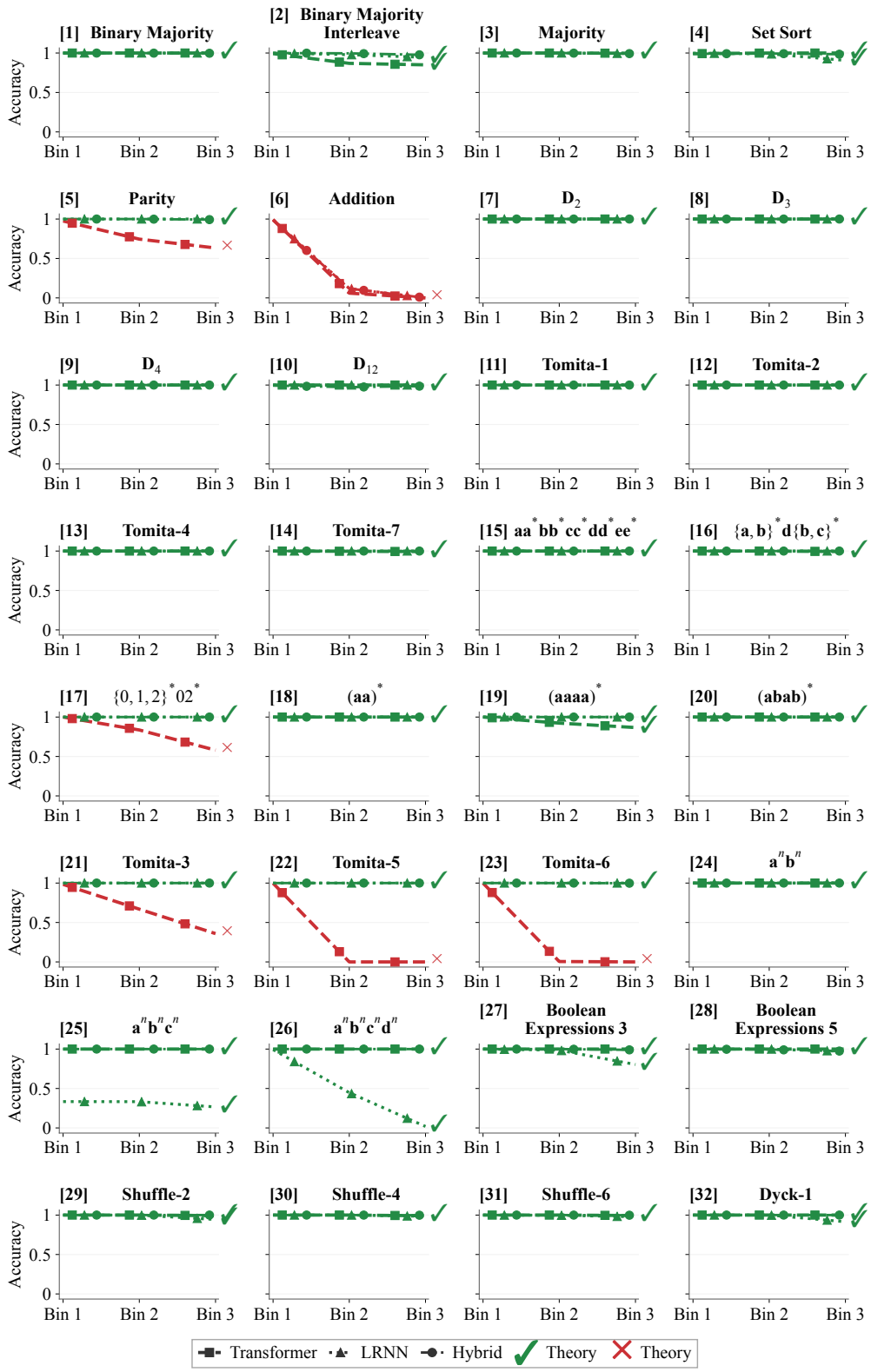


Figure 5: **Formal languages and algorithmic tasks.** Full version of Figure 4 in the main paper.

Training draws examples uniformly from that corpus. Following standard practice (Zhou et al., 2024; Huang et al., 2025a), absolute positional encodings are given random offsets during training, drawn uniformly from $[0, L_{\max} - \ell]$ for an instance of length ℓ and maximum evaluation length $L_{\max}$, so that the model can only rely on relative positions.

Optimization. All models are trained from scratch with HuggingFace's `Trainer`, using AdamW. The default budget is 30k steps with no warmup. Configurations with more than four Transformer or SSM layers, or more than two Hybrid motif repetitions, instead use 60k steps and 3000 linear warmup steps. Selective Copy, $a^n b^n$, $a^n b^n c^n$, $a^n b^n c^n d^n$, and Unique Suffix Copy were additionally trained for 120k steps, as training was more difficult on these structurally harder or scarce on data tasks. During training, every model is evaluated on each bin every 100 steps, giving at least 300 accuracy measurements per bin per 30k-step run.

Run selection. For each architecture-task combination we run a sweep over the cross product of hyperparameters in Table 6 for one seed. Once a hyperparameter combination achieves near perfect accuracy on the train distribution, as well as good length generalization ($>$0.9 accuracy on the last evaluation bin), we stop architecture sweeps. The best architectures are rerun on at least two more seeds to report averages on the top three runs with highest accuracies in their last bins. If no hyperparameter combination achieved $>$0.9 accuracy, we select the best-performing combination out of the ones available, and retrain it on the task with three more seeds. We report the average over seeds. If we achieved length generalization on a certain task on one architecture, we make sure to train the architectures not achieving length generalization on this task with at least as wide range of hyperparameters.

Model implementations. We use two implementation variants. The first builds on OLMo-core's `Llama-like` transformer (RMSNorm, SwiGLU feed-forward, pre-norm blocks). The model family only differs in the sequence mixer of each block: attention only (Transformer), Gated DeltaNet only (SSM), or the two interleaved (Hybrid). The Hybrid repeats a layer motif `sa` L times, where `s` denotes a recurrence block and `a` an attention block. We ran experiments on other layer motifs, but fell back to a simple recurrence-attention stack, as other layer stacks were harder to control. For recurrence layers we use Gated DeltaNet with channel-wise gating (`GDN2`), allowing negative eigenvalues in the transition (Grazzi et al., 2025), with a value expansion factor of 2 and head dimension d/h. SSMs themselves carry no positional encodings. We use a secondary implementation using the GPT2 head model of the `Transformers` library to run Transformers with absolute positional encodings.

Model	PE	Config	LR	ID	OOD 1	OOD 2
Digit by Digit Addition						
LRNN	none	1L 8d 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
S5 State Tracking						
LRNN	none	1L 16d 0.0dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**

Table 7: Average accuracies over top 3 seeds on additional algorithmic tasks. Runs are evaluated on in-distribution lengths 0–50 (ID) and are evaluated out of distribution on lengths 51–100 (OOD 1) and 101–150 (OOD 2). Accuracy is in percent and is bold at or above 98%. Model specifications are reported in *Config*, *PE*, and *LR*. The *config* has number of layers, the embedding dimension *d*, dropout rate *dr*, and training steps. *PE* denotes the positional encoding variant, and *LR* the learning rate. For hybrid models, *L* denotes the number of repetitions of the recurrence (`s`) and attention (`a`) layers (`sa` with *L*=2 corresponds to four total layers).

Model	PE	Config	LR	ID	OOD 1	OOD 2
Boolean Expressions 3						
LRNN	none	1L 8d 0dr 30k	10^{-3} / 10^{-4}	**100.0**	**98.6**	80.4
Hybrid	NoPE	2L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**98.9**
Transformer	APE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**99.9**	**98.8**
Transformer	RoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Boolean Expressions 5						
LRNN	none	2L 16d 0.0dr 30k	10^{-3}	**100.0**	**100.0**	97.1
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**99.4**	97.3
Transformer	APE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**99.2**
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**99.3**
Shuffle-2						
LRNN	none	1L 16d 0.1dr 30k	10^{-3}	**100.0**	**99.9**	94.4
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**99.9**	93.4
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-4}	**100.0**	**99.9**	97.4
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**99.3**
Shuffle-4						
LRNN	none	1L 16d 0.0dr 30k	10^{-4}	**100.0**	**100.0**	**98.4**
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**99.8**
Transformer	APE	2L 64d 2h 0.0dr 30k	10^{-4}	**100.0**	97.9	60.5
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**99.1**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-4}	**100.0**	**100.0**	97.5
Shuffle-6						
LRNN	none	1L 16d 0.0dr 30k	10^{-4}	**100.0**	**99.7**	97.8
Hybrid	NoPE	1L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**99.8**
Transformer	APE	2L 64d 2h 0dr 30k	10^{-4} / 10^{-3}	**100.0**	97.0	49.0
Transformer	RoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**99.4**
Transformer	NoPE	2L 64d 2h 0dr 30k	10^{-4} / 10^{-3}	**100.0**	**99.8**	77.2
Dyck-1						
LRNN	none	2L 16d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	91.5
Hybrid	NoPE	2L 64d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
$a^n b^n$						
LRNN	none	2L 16d 0dr 120k	10^{-2}	100.0	100.0	100.0
Hybrid	NoPE	1L 8d 2h 0.1dr `sa` 120k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
$a^n b^n c^n$						
LRNN	none	2L 64d 0.1dr 30k	10^{-2}	33.4	33.3	26.6
Hybrid	NoPE	1L 16d 2h 0.1dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
$a^n b^n c^n d^n$						
LRNN	none	2L 32d 0dr 30k	10^{-2}	**99.3**	44.7	1.9
Hybrid	NoPE	1L 16d 2h 0.1dr `sa` 120k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 64d 2h 0.1dr 30k	10^{-3}	**100.0**	79.8	57.1
Transformer	NoPE	2L 64d 2h 0.0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**

Table 8: Average accuracies over the top 3 seeds on the counter languages (1/2). Results are reported similar to Table 7

Model	PE	Config	LR	ID	OOD 1	OOD 2
Parity						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**99.2**
Transformer	RoPE	4L 16d 4h 0.1dr 60k	10^{-3}	91.2	62.4	51.5
LRNN	none	1L 64d 0.1dr 60k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 256d 2h 30k	10^{-4}	**100.0**	83.0	62.1
Transformer	NoPE	12L 768d 12h 60k	3×10^{-5}	97.5	74.5	63.3
Binary Majority						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**99.9**	**99.7**
Transformer	RoPE	1L 16d 4h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	2L 64d 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Interleaved Binary Majority						
Hybrid	NoPE	2L 64d 2h 0dr `sa` 30k	10^{-3}	**100.0**	**99.6**	97.6
Transformer	RoPE	4L 64d 2h 0.1dr 60k	10^{-3}	**99.2**	86.9	85.0
LRNN	none	4L 64d 0.1dr 60k	10^{-3}	**100.0**	97.9	94.5
Transformer	APE	2L 256d 4h 30k	10^{-4}	**100.0**	96.8	84.7
Transformer	NoPE	12L 768d 12h 60k	10^{-4}	96.2	68.8	39.5
Majority						
Hybrid	NoPE	1L 64d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**99.2**
Transformer	RoPE	1L 64d 4h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**99.7**
LRNN	none	2L 64d 0dr 30k	10^{-4}	**100.0**	**100.0**	**99.4**
Transformer	APE	1L 256d 2h 30k	10^{-3}	**100.0**	**99.6**	97.8
Transformer	NoPE	1L 64d 1h 30k	10^{-3}	**100.0**	**99.9**	**98.5**
Addition						
Hybrid	NoPE	4L 64d 2h 0dr `sa` 60k	10^{-3}	**98.2**	11.8	0.0
Transformer	RoPE	8L 256d 4h 0.1dr 60k	10^{-3}	**99.2**	6.0	0.0
LRNN	none	8L 64d 0dr 60k	10^{-3}	**99.3**	11.9	0.0
Transformer	APE	12L 768d 12h 60k	10^{-4}	**99.7**	26.8	0.0
Transformer	NoPE	12L 768d 12h 60k	10^{-4}	**99.5**	27.7	0.0
Unique Copy						
Hybrid	NoPE	1L 64d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**99.8**	**99.3**
Transformer	RoPE	4L 64d 1h 0dr 30k	10^{-3}	**100.0**	28.3	0.0
LRNN	none	2L 64d 0.1dr 30k	10^{-4}	**100.0**	85.7	6.9
Transformer	APE	2L 64d 1h 30k	10^{-3}	**100.0**	**99.9**	**98.7**
Transformer	NoPE	4L 256d 4h 30k	10^{-3}	**100.0**	18.0	0.0
Repeat Copy						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	96.9	6.4	0.0
Transformer	RoPE	4L 16d 4h 0dr 60k	10^{-3}	**100.0**	8.3	0.0
LRNN	none	4L 64d 0dr 60k	10^{-3}	**100.0**	32.8	0.0
Transformer	APE	4L 256d 4h 30k	10^{-3}	**100.0**	25.6	0.0
Transformer	NoPE	4L 256d 4h 30k	10^{-3}	**100.0**	28.0	0.0
Set Sort						
Transformer	NoPE	2L 64d 2h 0.1dr	10^{-3}	**99.2**	**100.0**	**100.0**
Transformer	APE	2L 64d 2h 0.1dr	10^{-4}	**98.7**	**100.0**	**100.0**
Transformer	RoPE	2L 64d 2h 0dr	10^{-3}	**99.9**	93.5	57.8
Hybrid	RoPE	2L 64d 2h 0.1dr `sa`	10^{-3}	**100.0**	**99.3**	82.2
Hybrid	NoPE	2L 64d 2h 0.1dr `sa`	10^{-3}	**99.0**	**99.4**	**98.4**
LRNN	none	4L 16d 0.1dr	10^{-3}	**99.8**	**99.0**	90.6

Table 9: Average accuracies on the algorithmic tasks. Results are reported as in Table 7.

Model	PE	Config	LR	ID	OOD 1	OOD 2
Multiplication						
Hybrid	NoPE	4L 256d 1h 0dr `sa` 60k	10^{-4}	55.8	2.0	0.7
Transformer	RoPE	8L 256d 2h 0dr 60k	10^{-3}	48.6	1.5	0.3
LRNN	none	8L 64d 0dr 60k	10^{-3}	55.6	1.9	0.6
Transformer	APE	4L 256d 4h 0dr 30k	10^{-3}	33.1	1.2	0.9
Transformer	NoPE	4L 256d 2h 0dr 30k	10^{-4}	31.1	1.5	1.0
Selective copy						
Hybrid	NoPE	2L 64d 4h 0.1dr `sa` 30k	10^{-3}	**98.7**	97.9	97.0
Transformer	RoPE	4L 64d 1h 0dr 30k	10^{-4}	**100.0**	87.7	63.9
LRNN	none	4L 16d 0dr 120k	10^{-3}	95.0	93.8	91.9
Transformer	APE	1L 256d 1h 0.1dr 120k	10^{-4}	18.6	18.1	15.9
Transformer	NoPE	1L 64d 1h 0dr 120k	10^{-3}	15.8	11.6	11.3
Unique Suffix Copy						
LRNN	none	2L 64d 0.1dr 120k	10^{-3}	**100.0**	80.5	9.9
Hybrid	NoPE	1L 64d 2h 0.1dr `saa` 30k,120k	10^{-3}	**100.0**	**99.8**	**99.2**
Transformer	APE	2L 256d 4h 0.1dr 120k	10^{-3}	93.0	62.3	55.1
Transformer	RoPE	4L 64d 2h 0.1dr 120k	10^{-4}	**100.0**	39.4	2.2
Transformer	NoPE	2L 64d 2h 0.0dr 120k	10^{-3}	17.4	0.0	0.0

Table 10: Average accuracies over top 3 seeds on additional tasks. Averages are taken over three seeds. Results are reported as in Table 7.

Model	PE	Config	LR	ID	OOD 1	OOD 2
D_2 2–50 / 51–100 / 101–150						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 1h 0.1dr 30k	10^{-4}	**100.0**	**100.0**	86.7
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
D_3 2–50 / 51–100 / 101–150						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 2h 0dr 30k	10^{-4}	97.3	69.3	21.6
LRNN	none	1L 16d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
D_4 2–100 / 101–200 / 201–300						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 64d 1h 0.1dr 30k	10^{-3}	**100.0**	**99.1**	**99.6**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
D_{12} 2–100 / 101–200 / 201–300						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-4}	**99.3**	97.3	**98.7**
Transformer	RoPE	2L 16d 4h 0dr 30k	10^{-4}	**99.9**	**99.1**	88.7
LRNN	none	4L 8d 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 128d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**

Table 11: Average accuracies over the top 3 seeds on the regular languages (1/3). Results are reported as in Table 7.

Model	PE	Config	LR	ID	OOD 1	OOD 2
Tomita 1	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 1h 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Tomita 2	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 1h 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	1L 16d 1h 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Tomita 3	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	4L 256d 2h 0.1dr 30k	10^{-4}	**98.3**	66.9	35.9
LRNN	none	4L 16d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	6L 128d 4h 30k	10^{-3}	60.5	7.7	0.0
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	21.1	0.0	0.0
Tomita 4	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 64d 1h 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 128d 2h 30k	10^{-3}	**100.0**	85.4	84.4
Transformer	NoPE	4L 64d 2h 30k	10^{-3}	**100.0**	16.5	0.0
Tomita 5	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	4L 256d 2h 0.1dr 30k	10^{-4}	**99.8**	0.0	0.0
LRNN	none	4L 16d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	6L 128d 4h 30k	10^{-3}	54.4	0.0	0.0
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	7.1	0.0	0.0
Tomita 6	1–100 / 101–150 / 151–200					
Hybrid	NoPE	2L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**99.8**
Transformer	RoPE	4L 256d 2h 0.1dr 30k	10^{-4}	**99.9**	0.5	0.0
LRNN	none	4L 16d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	6L 128d 4h 30k	10^{-3}	59.6	0.0	0.0
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	2.4	0.0	0.0
Tomita 7	1–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 16d 1h 0dr 30k	10^{-4}	**100.0**	**99.7**	**99.3**
LRNN	none	1L 8d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 128d 2h 30k	10^{-3}	**100.0**	94.2	93.2
Transformer	NoPE	4L 64d 2h 30k	10^{-3}	**100.0**	87.5	76.2

Table 12: Average accuracies over the top 3 seeds on the regular languages (2/3). Results are reported as in Table 7.

Model	PE	Config	LR	ID	OOD 1	OOD 2
$aa^*bb^*cc^*dd^*ee^*$	5–200 / 201–300 / 301–400					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 1h 0dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 64d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	6L 64d 4h 30k	10^{-4}	**100.0**	**100.0**	**100.0**
$(ab)^*d(bc)^*$	2–50 / 51–100 / 101–150					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 1h 0dr 30k	10^{-4}	**100.0**	**100.0**	**99.3**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 64d 4h 30k	10^{-4}	**100.0**	**99.8**	**98.5**
Transformer	NoPE	6L 64d 4h 30k	10^{-4}	**100.0**	**99.8**	**98.5**
$(012)^*(0\|2)^*$	2–50 / 51–100 / 101–150					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 64d 1h 0.1dr 30k	10^{-3}	**100.0**	83.6	58.0
LRNN	none	1L 8d 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	APE	6L 128d 4h 30k	10^{-3}	**99.7**	24.4	1.2
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	18.2	11.2	0.0
$(aa)^*$	2–100 / 101–150 / 151–200					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	1L 16d 4h 0dr 30k	10^{-3}	**100.0**	0.0	0.0
LRNN	none	1L 16d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 128d 4h 40k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	**100.0**	0.0	0.0
$(abab)^*$	4–200 / 201–400 / 401–600					
Hybrid	NoPE	1L 16d 1h 0dr `sa` 30k	10^{-3}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	2L 16d 4h 0.1dr 30k	10^{-3}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 8d 0.1dr 30k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	APE	4L 128d 4h 40k	10^{-4}	**100.0**	89.4	82.3
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	**100.0**	0.0	0.0
(aaaa)*						
Hybrid	NoPE	1L 16d 1h 0dr `sa` 60k	10^{-4}	**100.0**	**100.0**	**100.0**
LRNN	none	1L 16d 4h 60k	10^{-4}	**100.0**	**100.0**	**100.0**
Transformer	RoPE	6L 256d 4h 60k	10^{-4}	**100.0**	0.0	0.0
Transformer	APE	4L 128d 4h 40k	10^{-4}	**100.0**	92.3	86.7
Transformer	NoPE	6L 256d 4h 60k	10^{-4}	**100.0**	0.0	0.0

Table 13: Average accuracies over the top 3 seeds on the regular languages (3/3). Results are reported as in Table 7.

E AI USAGE

We have used modern LLMs to help in our project in different ways. We used LLM based coding assistants to help with understanding the code from different codebases, from which we sourced the relevant datasets. We further used LLM based coding assistants to help in homogenizing the code from these different codebases and then running them all through the same pipeline. The different plots in the paper were also made with such agents. We also used ChatGPT to brainstorm ideas about naming our different RASP variants, and also brainstorm other aspects of the project, including what seems to be gaps in the overall narrative. LLMs were also used in helping format the various tables in our paper better, and to ensure that none of the tables violate any of the margin requirements. Crucially LLMs were not used for drafting the paper, but they were used in catching typos and helping find phrasings which were unclear. The phrasing corrections however were subsequently then also done by a human. Finally LLMs were also used to in pointing out papers from the literature, we had missed addressing in our results.