RECURSIVE LANGUAGE MODELS GENERALIZE OUT OF DOMAIN

Anonymous authors
Paper under double-blind review

ABSTRACT

We study when limiting what a language model can see improves learning. We compare standard CoT, the more general learner that reads the full trace, with recursive language models, which restricts itself by solving each subtask in an isolated context. In-distribution, this generality comes for free: CoT can efficiently simulate the recursive rule, so the IID generalization guarantee changes only by a constant factor, and recursion does not offer much. But out of domain, CoT can fit training by relying on context outside the current subtask, i.e. a shortcut that breaks once those tokens change; recursive context isolation rules out this failure mode. Even though CoT's class still covers the recursive rule, simplicity bias picks the shortcut over the truth. Thus, to go beyond distributional accuracy and truly reason, covering the right rule is not enough—in contrast to classical learning theory.

1 INTRODUCTION

A central lesson of machine learning is to favor generality over hand-built specificity (Sutton, 2019): use a broad, scalable model class and let data pick the rule. This is statistically cheap in IID learning, where the usual guarantees pay for a simple rule that fits the data, not for every rule the class could express (Rissanen, 1978; Grünwald, 2007; Shalev-Shwartz & Ben-David, 2014). This paper asks where that argument stops. The argument is distributional at heart, but language models are increasingly expected to reason—to learn the rule itself, and able to answer correctly on any instance, including ones outside training. This is the out-of-domain (OOD) generalization question.

We make this tension concrete in language modeling. In standard chain-of-thought (CoT) reasoning (Nye et al., 2021; Wei et al., 2022), every subproblem and every intermediate step is written into one growing sequence. A recursive (language) model (RM) (Yang et al., 2026; Zhang et al., 2025a) instead starts a new isolated sequence for each subproblem, runs the same predictor there, and passes only the answer back to the parent. A child can open further subproblems, producing a stack of contexts (Figure 1). The same design principle—isolated context instead of the full history—is widely used in current LLM systems (Rajasekaran et al., 2025; OpenAI, 2026). The immediate benefit is computational: RM reads a shorter sequence at each step and can handle computations whose flattened CoT trace would exceed the context budget.

But the learnability question is subtler. Setting aside the context-length and compute benefits, does RM learn better because it hides the parts of the complete sequence that are irrelevant to the current subproblem? The answer depends on the kind of generalization. On i.i.d. samples from the training generator, RM needs fewer samples to reach high accuracy but CoT catches up with enough data (§ 6.1 and fig. 3); in distribution, CoT is not punished for seeing too much. Out of domain, the conclusion reverses: CoT collapses on traces only modestly longer than training, while RM stays accurate across much longer traces (§ 6.2 and table 1).

In § 4, we formalize the in-distribution result via the usual generality-over-specificity argument. CoT is the more general learner: it reads the complete sequence rather than enforcing the context isolation that RM builds in. This generality is cheap in distribution because the recursive rule is compactly contained inside CoT: a CoT Transformer can reconstruct the sequence RM would have seen and apply the same next-token rule, using only constant overhead in depth and parameter count. The corresponding IID upper bound changes only by a constant-factor overhead in the description-length term.

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.

OOD is where specificity starts to matter. The same generality that lets CoT contain the recursive rule also lets its attention range over the whole flattened trace at each prediction step. Tokens outside the current subproblem can form easy-to-read patterns—nearby intermediate values, repeated formatting, regularities in where answers appear—that are predictive in the training domain without being part of the subproblem's actual rule. For example, an inner subproblem may return a value that remains visible in the flattened trace and happens to equal the final answer on the training domain, so CoT can learn to copy it (§ 6.3 and table 2). RM's context isolation hides such cues by construction.

§ 5 formalizes this danger via the Minimum Description Length (MDL) view of learning (Rissanen, 1978; Grünwald, 2007). The simplicity bias that makes generality cheap in distribution does not distinguish the intended mechanism from a shorter rule over the complete sequence that merely fits training—a shortcut. The desired counterproperty is invariance (Peters et al., 2016; Arjovsky et al., 2019): if the correct answer for a subproblem depends only on the subproblem's own description, then changing the surrounding sequence should not change the prediction. RM satisfies this by construction; CoT can represent the invariant rule, but can also represent shorter non-invariant rules that fit the same training examples. We prove that a CoT model can generalize IID perfectly over complete traces and still fail when asked to solve that subproblem alone—it has learned how the answer is exposed in the trace, not how to solve the subproblem.

This suggests that out-of-domain reasoning needs specificity, not just coverage—the bitter lesson. Our contribution is to show how the classical simplicity-based view of learning plays out differently in IID prediction and OOD reasoning in language models and agentic systems. The simplicity bias that makes generality cheap in distribution can favor shortcuts out of domain. Context isolation hides the external cues that support these shortcuts, offering a partial explanation for its generalization benefits beyond computational efficiency. Further related work is discussed in § A.

2 Background: Description Length and MDL

A broad, scalable function class can be enough, as long as it *covers* a good rule and learning has a bias toward finding a simple one.

Definition 1 (Complexity Measure)**.** *A* complexity measure *is a function ℓ assigning each predictor f a score $\ell(f) \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$ such that $\sum_f 2^{-\ell(f)} \leq 1$. We write $|f|_\ell := \ell(f)$.*

The term *description length* is justified by the Kraft condition in the definition: it is the standard prefix-code condition that lets $|f|_\ell$ be read as the number of bits needed to describe f. Further examples and the code interpretation are given in § B.

Let Ω be the input space and let $\mathcal{X} \subseteq \Omega$ be the observed inputs. For two predictors f_1, f_2, write $f_1 \equiv_{\mathcal{X}} f_2$ if $f_1(\mathbf{x}) = f_2(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$.

Among all predictors consistent with the observed data, MDL selects one with the shortest description.

Definition 2 (MDL Learner)**.** *Given a complexity measure ℓ, target f^*, and observed inputs $\mathcal{X}$, the* MDL learner *returns*

$$\mathsf{MDL}_\ell(\mathcal{X}, f^*) = \arg \min_{f:\, f \equiv_{\mathcal{X}} f^*} |f|_\ell. \tag{1}$$

We assume a finite-length consistent predictor exists and use fixed deterministic tie-breaking.

The standard realizable MDL bound and its proof are in § B.2. Thus IID learning pays for a short consistent rule, not for every rule with finite description length (§ B.4). Two predictors can both fit the training data and receive comparable IID guarantees, while disagreeing elsewhere.

3 Learning Recursive Language Models

Recursive language models are our test case for this tension.

3.1 Autoregressive CoT and Recursive Models

Autoregressive generation. Let Σ be a finite vocabulary and set $\Omega := \Sigma^*$. We write ϵ for the empty sequence and $\|$ for concatenation. For a parameter value θ, a next-token predictor $f_\theta : \Omega \to \Sigma$ maps

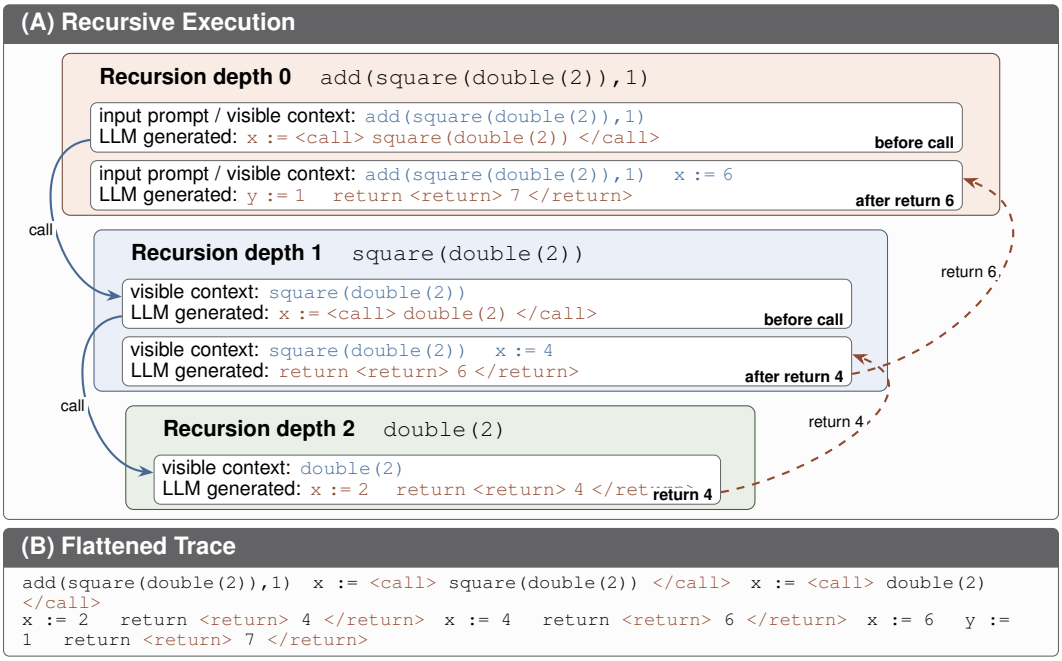

Figure 1: **Recursive expression evaluation.** The example evaluates $\texttt{add}(\texttt{square}(\texttt{double}(2)), 1)$ with arithmetic modulo 10. The same computation can be viewed either as nested recursive calls or as one flattened trace.

a sequence to the next token. Starting from a prompt $\mathbf{x}_0 \in \Omega$, autoregressive generation produces $\mathbf{x}_{t+1} = \mathbf{x}_t \parallel f_\theta(\mathbf{x}_t)$, where $\mathbf{x}_t$ is the sequence after t prediction steps. In chain-of-thought (CoT) reasoning (Nye et al., 2021; Wei et al., 2022), the complete sequence $\mathbf{x}_t$ is visible to the predictor.

Recursive generation. The *recursive model* (Yang et al., 2026; Zhang et al., 2025a) changes what the model reads at each step. When it reaches a subproblem, it starts a new instance of the same predictor, with the same weights but a new isolated context initialized by that subproblem. This child instance may solve the subproblem directly or make further recursive calls. When it returns, only the answer is passed back to the parent context; the child's scratch work is recorded in the complete sequence but is not copied into the parent.

Equivalently, the recursive model maintains a stack of *frames*. The top frame is the *active frame*, and it is the only text shown to the running model instance. Reserve four distinct control tokens $\Sigma_{\text{ctrl}} := \{\langle\texttt{call}\rangle, \langle/\texttt{call}\rangle, \langle\texttt{return}\rangle, \langle/\texttt{return}\rangle\} \subseteq \Sigma$, and write $\Sigma_{\text{ord}} := \Sigma \setminus \Sigma_{\text{ctrl}}$. A call block $\langle\texttt{call}\rangle\, \mathbf{q}\, \langle/\texttt{call}\rangle$ with $\mathbf{q} \in \Sigma_{\text{ord}}^*$ pushes a child frame initialized with $\mathbf{q}$; a return block $\langle\texttt{return}\rangle\, \mathbf{a}\, \langle/\texttt{return}\rangle$ with $\mathbf{a} \in \Sigma_{\text{ord}}^*$ pops the child frame and appends $\mathbf{a}$ to its parent. All other tokens append to the top frame.

Observation-function view. The stack view leaves two objects to keep distinct: the complete sequence accumulated token by token, and the context shown before the next prediction. In Figure 1, panel (B) is the complete sequence, while panel (A) shows the recursive call structure and the frame visible to the predictor. We encode the visible context as a function of the complete sequence. Let $\Pi(\mathbf{x})$ be the context shown when the accumulated sequence is $\mathbf{x}$. A visible-context predictor $g_\theta : \Omega \to \Sigma$ generates $\mathbf{x}_{t+1} = \mathbf{x}_t \parallel g_\theta(\Pi(\mathbf{x}_t))$. The next token is still appended to the complete sequence; Π only controls what the predictor reads before choosing it.

Definition 3 (Observation Function)**.** *An* observation function *is an idempotent map $\Pi : \Omega \to \Omega$ that returns the context visible to the predictor: $\Pi \circ \Pi = \Pi$. Idempotence says that once a sequence has been reduced to its visible context, observing it again does not reveal additional text.*

The identity map gives ordinary CoT. For recursive models, the relevant observation function is the active-frame map Π_{rec}. Given a complete sequence $\mathbf{x}$, Π_{rec} replays the call and return tokens

and returns the top frame; § D.1 gives this replay procedure. Since the output is already one frame, $\Pi_{\mathrm{rec}} \circ \Pi_{\mathrm{rec}} = \Pi_{\mathrm{rec}}$.

3.2 Learning with CoT vs Recursive Models

The recursive model gives a natural training recipe: use the active frame as input for each next-token prediction. The question is whether the active-frame input makes the rule easier to learn, or whether CoT can learn the same next-token behavior from complete sequences.

Write g^* for the target rule on active frames, and $f^* = g^* \circ \Pi_{\mathrm{rec}}$ for the induced target on complete sequences. Let $\mathcal{X}$ be the complete sequences observed in recursive model runs. For each $\mathbf{x} \in \mathcal{X}$, CoT trains on $(\mathbf{x}, f^*(\mathbf{x}))$; the recursive model trains on $(\Pi_{\mathrm{rec}}(\mathbf{x}), f^*(\mathbf{x}))$, equivalently $(\Pi_{\mathrm{rec}}(\mathbf{x}), g^*(\Pi_{\mathrm{rec}}(\mathbf{x})))$. Thus both learners use the same recursive runs and next-token labels; only the input shown to the learner changes.

MDL comparison. For the MDL comparison, fix an arbitrary observation function Π and assume the complete-sequence target has the form $f^* = g^* \circ \Pi$. We use one base complexity measure ℓ for next-token rules; the difference between the learners should come from which part of the sequence the predictor reads, not from changing what counts as a simple rule. CoT applies ℓ directly to complete-sequence rules f. The recursive model first searches for a local rule g and then uses $g \circ \Pi$ on complete sequences. Equivalently, it is standard MDL on complete-sequence rules under the following lifted complexity measure.

Definition 4 (Lifted Complexity Measure)**.** *For a complexity measure ℓ and observation function Π, define the lifted complexity measure ℓ^{Π} by*

$$|f|_{\ell^{\Pi}} := \min_{g:\, f = g \circ \Pi} |g|_{\ell}, \tag{2}$$

with value $+\infty$ if no such g exists.

The Kraft verification is in § B.5. Lifted MDL minimizes $|g|_{\ell}$; this is different from minimizing the original length $|g \circ \Pi|_{\ell}$ of the composed complete-sequence rule.

Proposition 1 (Local Fitting as Lifted MDL)**.** *Let Π be an observation function, let $f^* = g^* \circ \Pi$, and write $\Pi(\mathcal{X}) := \{\Pi(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$. If deterministic tie-breaking is chosen consistently on the two minimization problems, then*

$$\mathsf{MDL}_{\ell^{\Pi}}(\mathcal{X}, f^*) = \mathsf{MDL}_{\ell}(\Pi(\mathcal{X}), g^*) \circ \Pi. \tag{3}$$

Thus MDL under the lifted measure is the same as first choosing the shortest local rule on the visible contexts $\Pi(\mathcal{X})$, then composing it with Π.

Below, $f, \hat{f}$ read complete sequences and $g, \hat{g}$ read visible contexts. CoT returns $\hat{f} := \mathsf{MDL}_{\ell}(\mathcal{X}, f^*)$; lifted MDL returns $\hat{f}_{\Pi} := \mathsf{MDL}_{\ell^{\Pi}}(\mathcal{X}, f^*) = \hat{g} \circ \Pi$, with consistent tie-breaking.

4 IID Generalization: Comparing CoT and Recursive Models

When training and test sequences come from the same distribution, does recursion make learning easier simply by restricting the predictor to the visible context rather than the complete sequence? We isolate this IID question by first showing that CoT can simulate the recursive view with constant depth and constant-factor parameter overhead, then translating the simulation into the MDL comparison of § 2.

Expressiveness. Let $\mathcal{A}$ be a Transformer architecture formally defined in § C, with depth $L_{\mathcal{A}}$ and parameter count $W_{\mathcal{A}}$.[1] The corresponding next-token predictor class is $\mathcal{F}_{\mathcal{A}}$.

Theorem 2 (CoT Simulation of the Recursive Model)**.** *For every such Transformer architecture $\mathcal{A}$, there exists another architecture $\mathcal{A}'$ with $L_{\mathcal{A}'} = L_{\mathcal{A}} + O(1)$ and $W_{\mathcal{A}'} = O(W_{\mathcal{A}})$. For every choice of visible-context recursive rule $g \in \mathcal{F}_{\mathcal{A}}$, there exists a CoT simulator $f \in \mathcal{F}_{\mathcal{A}'}$ such that $f(\mathbf{x}) = g(\Pi_{\mathrm{rec}}(\mathbf{x}))$ for every complete sequence $\mathbf{x} \in \Omega$ that can arise during recursive generation (see Definition 7 for the formal definition).*

[1] We assume exact arithmetic, reindexable positional features, causal rightmost-hard attention (Hahn, 2020; Merrill et al., 2022) and product-gated FFNs (Shazeer, 2020).

The proof is deferred to § D. A token inside an unfinished call/return block has two roles: it belongs to the current unfinished suffix, and after the block closes it belongs to its eventual frame. The construction maintains an owner state and a pending state at each position, simulates the original layers on both in parallel, and selects the state for the active frame. The simulator architecture $\mathcal{A}'$ is obtained from $\mathcal{A}$ by a fixed computable construction.

Description-length comparison. We charge for describing $\mathcal{A}$, and then charge b bits for each independently stored learned parameter, taking the shortest such encoding for each rule. For the recursive model, the corresponding description language is the lifted language $\ell^{\Pi_{\mathrm{rec}}}$. The full coding convention is in § E.

Corollary 3 (Comparable Description Length for CoT)**.** *For every complete-sequence predictor f with finite $|f|_{\ell^{\Pi_{\mathrm{rec}}}}$, there is a CoT predictor $\tilde{f}$ that agrees with f on the legal recursive-prefix domain considered in Theorem 2, and constants C, C_0 independent of the task and training sample such that*

$$|\tilde{f}|_{\ell} \le C\,|f|_{\ell^{\Pi_{\mathrm{rec}}}} + C_0. \tag{4}$$

The fixed architecture translation adds only $O(1)$ code bits, and the learned parameters grow by only a constant factor (Lemma 17).

IID Generalization. Let $\mathcal{D}$ be any distribution over complete sequences. Fix a visible-context recursive rule $g \in \mathcal{F}_{\mathcal{A}}$, and let $f := g \circ \Pi_{\mathrm{rec}}$ be the induced complete-sequence target. For any predictor $h : \Omega \to \Sigma$, measure error against this target by $\mathcal{L}_{\mathcal{D}}(h) := \Pr_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f(\mathbf{x})]$.

Theorem 4 (IID Generalization Under CoT Simulation)**.** *Assume $|f|_{\ell^{\Pi_{\mathrm{rec}}}} < +\infty$ and that a draw $\mathbf{x} \sim \mathcal{D}$ can arise during recursive generation almost surely. Let $\mathcal{X} \sim \mathcal{D}^m$ be the training set of sequences. Define*

$$\hat{f} := \mathsf{MDL}_{\ell}(\mathcal{X}, f), \qquad \hat{f}_{\Pi_{\mathrm{rec}}} := \mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f). \tag{5}$$

Then, for any $\delta \in (0,1)$, with probability at least $1-\delta$ over the draw of $\mathcal{X}$,

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) \le \frac{\big(C\,|f|_{\ell^{\Pi_{\mathrm{rec}}}} + C_0\big)\ln 2 + \ln(2/\delta)}{m}, \qquad \mathcal{L}_{\mathcal{D}}(\hat{f}_{\Pi_{\mathrm{rec}}}) \le \frac{|f|_{\ell^{\Pi_{\mathrm{rec}}}}\ln 2 + \ln(2/\delta)}{m}. \tag{6}$$

The proof applies the realizable MDL bound twice: once under the lifted description language $\ell^{\Pi_{\mathrm{rec}}}$ and once under the CoT description language ℓ. The resulting sufficient sample bounds differ only by the simulation overhead in the description-length term (§ E).

5 OOD Generalization: Invariance and Risks of Shortcuts

5.1 Why OOD Is Different from IID

IID learning asks whether the selected rule predicts well under the same distribution that produced the training data. Let $\mathcal{X} \subseteq \Omega$ denote the training set of complete sequences, with target values given by a rule $f^* : \Omega \to \Sigma$. We focus on consistent learners: the learned predictor agrees with f^* on $\mathcal{X}$, and we ask how it behaves beyond $\mathcal{X}$.

Risks of simplicity bias. Here simplicity bias plays a different role. In IID learning, a short rule that fits the training data receives a good average-case guarantee under the same distribution. Out of domain, the shortest fitting rule may instead use a pattern specific to the training domain rather than the mechanism we meant to learn. We call such a rule a shortcut: a predictor $\hat{r}$ with $\hat{r} \equiv_{\mathcal{X}} f^*$ and $|\hat{r}|_{\ell} < |f^*|_{\ell}$. From the observations alone, $\hat{r}$ is indistinguishable from the intended rule, so MDL may choose it. The difference appears only when we evaluate on inputs where the training-domain pattern no longer tracks f^*.

Figure 2 illustrates a recursive-trace shortcut: a returned value outside the active frame can agree with the answer during training and disagree after the surrounding computation changes. Two further examples involving strings and graphs are given in § H.

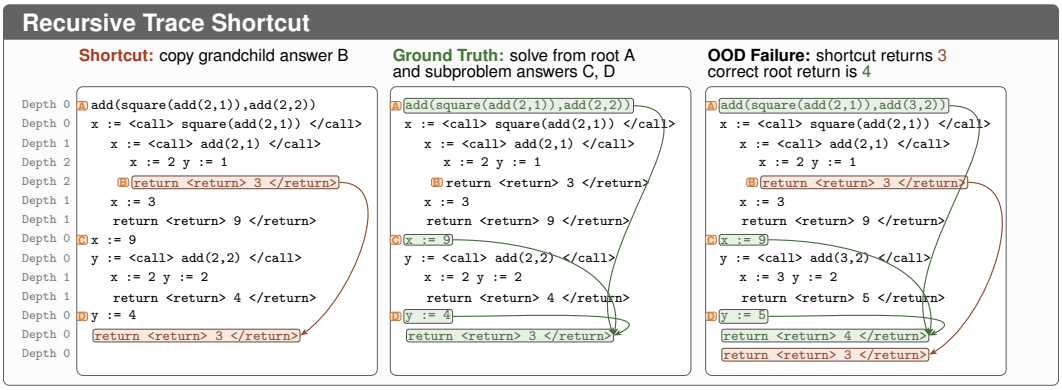

Figure 2: **A recursive-trace shortcut.** A simpler rule fits the training domain but fails once the training-domain correlation is broken. In recursive computation, reading the complete sequence enables shortcuts that use context irrelevant to the local computation.

5.2 Invariance Under an Observation Function

The same examples also suggest the fix: restrict what the predictor can read. If the predictor only receives the information that the target actually uses, then correlations in the surrounding complete sequence cannot affect its answer.

Fix an idempotent observation function $\Pi : \Omega \to \Omega$ (so $\Pi(\Pi(\mathbf{x})) = \Pi(\mathbf{x})$) and a local target rule g on visible contexts $\Pi(\mathbf{x})$. Viewed as a rule on complete sequences, the same behavior is $f(\mathbf{x}) = g(\Pi(\mathbf{x}))$. The simple observation is that any predictor that only reads $\Pi(\mathbf{x})$ must give the same answer on sequences with the same visible context. We formalize this as invariance.

Definition 5 (Invariance). *Let $\Pi : \Omega \to \Omega$ be an arbitrary observation function and let $\mathcal{X} \subseteq \Omega$ be the training set. We say a rule $p : \Omega \to \Sigma$ is* Π-invariant on the training set $\mathcal{X}$ *if, for all $\mathbf{x} \in \mathcal{X}$ and $\mathbf{x}_{\text{te}} \in \Omega$, if $\Pi(\mathbf{x}) = \Pi(\mathbf{x}_{\text{te}})$, then $p(\mathbf{x}) = p(\mathbf{x}_{\text{te}})$.*

Any local rule q used through Π is Π-invariant by construction, because its complete-sequence prediction has the form $q(\Pi(\mathbf{x}))$. We next name the inputs that reuse a training visible context.

Definition 6 (Equivalent set). *Fix an observation function $\Pi : \Omega \to \Omega$. Two complete sequences $\mathbf{x}, \mathbf{x}_{\text{te}} \in \Omega$ are* Π-equivalent *if $\Pi(\mathbf{x}) = \Pi(\mathbf{x}_{\text{te}})$. For a training set $\mathcal{X} \subseteq \Omega$, its* Π-equivalent set *is*

$$\text{Eq}_\Pi(\mathcal{X}) := \{\mathbf{x}_{\text{te}} \in \Omega : \exists \mathbf{x} \in \mathcal{X} \text{ such that } \Pi(\mathbf{x}_{\text{te}}) = \Pi(\mathbf{x})\} = \Pi^{-1}(\Pi(\mathcal{X})). \tag{7}$$

It contains exactly the complete sequences whose visible context has appeared in $\mathcal{X}$, and in particular $\mathcal{X} \subseteq \text{Eq}_\Pi(\mathcal{X}) \subseteq \Omega$.

Invariance gives the following transfer guarantee.

Proposition 5 (OOD Generalization to the Equivalent Set). *Suppose a predictor $\hat{f}$ is* consistent with the target f on the training set $\mathcal{X}$ *(i.e., $\hat{f}(\mathbf{x}) = f(\mathbf{x}) = g(\Pi(\mathbf{x}))$ for all $\mathbf{x} \in \mathcal{X}$) and* Π-invariant on $\mathcal{X}$. *Then $\hat{f}$ remains correct throughout the equivalent set: $\hat{f}(\mathbf{x}_{\text{te}}) = f(\mathbf{x}_{\text{te}}) = g(\Pi(\mathbf{x}_{\text{te}}))$ for all $\mathbf{x}_{\text{te}} \in \text{Eq}_\Pi(\mathcal{X})$.*

Thus, for a Π-invariant predictor, covering visible contexts in $\mathcal{X}$ gives correctness on new complete sequences that reuse those contexts. For recursive models, once a subproblem's active frame has appeared in training, changing the parent problem, recursion depth, or other surrounding tokens does not change the predictor's behavior on that subproblem. The guarantee does not cover genuinely new subproblems outside $\Pi(\mathcal{X})$.

5.3 Shortcuts in CoT

CoT reads the complete sequence $\mathbf{x}$, so in recursive traces it can attend to tokens outside the active frame $\Pi_{\text{rec}}(\mathbf{x})$. These tokens can support shortcuts: stack depth, sequence length, repeated tokens,

or a value that appears elsewhere in the complete sequence. For example, a value returned by one call may appear outside the active frame used for another prediction. CoT can copy that value on the training inputs instead of computing the answer from the active frame; when the surrounding complete sequence changes, the copy rule can fail. Figure 2 illustrates this shortcut in recursive computation.

Risks of Shortcuts for OOD Generalization. Let $f = g \circ \Pi$ and $\hat{f}_\Pi := \mathsf{MDL}_{\ell^\Pi}(\mathcal{X}, f)$. By Proposition 1, $\hat{f}_\Pi = \hat{g} \circ \Pi$ for a shortest local rule, and $|\hat{f}_\Pi|_{\ell^\Pi} = |\hat{g}|_\ell$. A strict shortcut gap means a consistent CoT rule is shorter than this local rule. It forces failure even on a familiar visible context presented by itself.

Theorem 6 (Shortcut Forces Non-Invariance)**.** *Let ℓ, $\mathcal{X} \subseteq \Omega$, and an idempotent observation function $\Pi : \Omega \to \Omega$ be arbitrary. For any visible-context target rule $g : \Omega \to \Sigma$, set $f := g \circ \Pi$. Let $\hat{f}_\Pi := \mathsf{MDL}_{\ell^\Pi}(\mathcal{X}, f)$ be well defined, and let $\hat{f}$ be any finite-length predictor satisfying $\hat{f} \equiv_\mathcal{X} f$. If*

$$|\hat{f}|_\ell < |\hat{f}_\Pi|_{\ell^\Pi}, \tag{8}$$

then the CoT predictor $\hat{f}$ is not Π-invariant on $\mathcal{X}$. That is, there exist $\mathbf{x} \in \mathcal{X}$ and $\mathbf{x}_{\mathrm{te}} \in \Pi(\mathcal{X}) \subseteq \Omega$ such that

$$\Pi(\mathbf{x}) = \Pi(\mathbf{x}_{\mathrm{te}}), \qquad \hat{f}(\mathbf{x}_{\mathrm{te}}) \neq \hat{f}(\mathbf{x}) = g(\Pi(\mathbf{x})). \tag{9}$$

Proof. Suppose, toward a contradiction, that $\hat{f}$ is Π-invariant on $\mathcal{X}$. Then, for every $\mathbf{x} \in \mathcal{X}$, $(\hat{f} \circ \Pi)(\mathbf{x}) = \hat{f}(\Pi(\mathbf{x})) = \hat{f}(\mathbf{x}) = f(\mathbf{x})$, so $\hat{f} \circ \Pi$ is feasible for $\mathsf{MDL}_{\ell^\Pi}(\mathcal{X}, f)$. By the definition of the lifted length, the same rule $\hat{f}$ can be used after applying Π to represent $\hat{f} \circ \Pi$, hence $|\hat{f} \circ \Pi|_{\ell^\Pi} \leq |\hat{f}|_\ell$. Since $\hat{f}_\Pi$ is the ℓ^Π-MDL solution,

$$|\hat{f}_\Pi|_{\ell^\Pi} \leq |\hat{f} \circ \Pi|_{\ell^\Pi} \leq |\hat{f}|_\ell, \tag{10}$$

contradicting the shortcut gap. Therefore there exists $\mathbf{x} \in \mathcal{X}$ such that $\hat{f}(\Pi(\mathbf{x})) \neq \hat{f}(\mathbf{x})$. Set $\mathbf{x}_{\mathrm{te}} := \Pi(\mathbf{x})$. Idempotence gives $\Pi(\mathbf{x}_{\mathrm{te}}) = \Pi(\mathbf{x})$, and consistency gives $\hat{f}(\mathbf{x}) = f(\mathbf{x}) = g(\Pi(\mathbf{x}))$. Thus $\mathbf{x}_{\mathrm{te}} \in \Pi(\mathcal{X})$ is the claimed non-invariance witness. □

The strict gap is sufficient, not necessary; the closure case and a counterexample to necessity are discussed in § F.1 and proposition 18.

Specializing to recursive models, the visible context is the active frame $\Pi_{\mathrm{rec}}(\mathbf{x})$, a subproblem that already appeared inside a complete sequence in the training set. The learned CoT predictor can answer correctly when the subproblem sits inside that sequence, but fail when the surrounding context changes, including when the same subproblem is presented on its own. **CoT has learned the subproblem as a pattern tied to the surrounding complete sequence, not as a context-independent local computation that can be reused when the surrounding sequence changes.**

6 Experiments

We use symbolic expression evaluation as a controlled testbed (Xu et al., 2026). The task is simple enough that the intended computation is unambiguous: each expression has an exact modulo-10 or Boolean answer. The generator lets us vary the amount of IID training data, the executed trace length, the recursion depth, and whether a shortcut is visible in the full trace.

Expressions are sampled from a small typed library with three kinds of functions. Primitive functions evaluate in one step; composite functions are named definitions that open further calls; and tail-recursive functions are named definitions that can call themselves with updated arguments. For example, `sum_of_squares`(a, b) expands into `add`(`square`(a), `square`(b)); the tail-recursive definitions and a worked trace are given in § G.1.

The sampler chooses a root function and recursively fills its arguments with literals or sub-expressions, enforcing type consistency and a bounded depth in the written expression. The bound is on the written expression, not on the executed trace: composite expansion and tail recursion can make a short root problem unfold into a long trace or a deep recursion stack.

Both learners use the same 6-layer Transformer backbone and are compared on the same underlying expression pools within each experiment. The primary task metric is final-answer accuracy; the IID experiment also reports per-token test accuracy to diagnose how well each model fits the local next-token objective. Training and checkpoint protocols are in §§ G.2 to G.5.

The experiments then ask three questions. How much IID data is needed to learn new recursive compositions from the same generator (§ 6.1)? What happens when test expressions unfold into longer traces or deeper recursion stacks than the training expressions (§ 6.2)? When CoT fails, do its predictions follow trace-level shortcuts that RM cannot observe (§ 6.3)?

6.1 IID Compositional Generalization

Setup. For each training-set size N, RM and CoT train on the same nested set of expressions from the generator and are evaluated on held-out expressions from the same distribution. Checkpoints are selected by IID validation performance; the full protocol is in § G.3.

Results. RM learns the final answer with fewer samples in the middle of the sweep, but CoT reaches the same near-perfect regime once the training set is large enough; on per-token accuracy, CoT is slightly higher at large N (Figure 3). The large gaps below appear only after the test distribution changes.

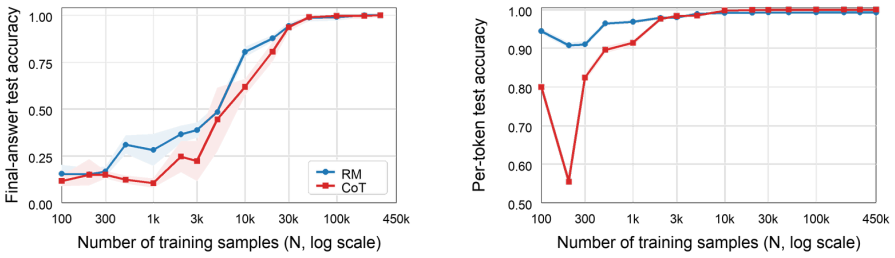

Figure 3: IID sample complexity: final-answer and per-token test accuracy.

6.2 Length and Depth Generalization

Setup. We use the same expression generator but change which executions are allowed in training. The length split orders examples by executed trace length: the number of tokens in the executed call/return trace, not the length of the written expression. The depth split orders examples by maximum recursion depth. For a threshold L or K, training examples lie below the threshold and OOD examples lie above it; held-out below-threshold examples give the IID check. Details are in § G.4.

Results. Table 1 separates the below-threshold and above-threshold bins. Below the threshold, both learners are nearly perfect on held-out examples. Above the threshold, CoT degrades sharply with longer traces and steadily with deeper stacks. RM remains high across the depth shift and most of the length shift. This matches the invariance picture from § 5: many OOD executions still reuse locally familiar subproblems for RM, while the flattened CoT trace has moved outside the training range. The largest RM drop occurs in the farthest length bin, where some active frames themselves become longer than those seen in training.

6.3 Shortcut Behavior

Setup and metric. The length/depth experiments show large CoT errors, but they do not by themselves identify what CoT learned. We therefore construct train/OOD pairs with a planted shortcut signal S. During training, S agrees with the answer, so a predictor can fit the data by using S instead of the intended computation. At OOD evaluation, the task format is unchanged but the coupling between S and the answer is broken. The signal is visible in the flattened CoT trace but hidden from RM's active frame. We report accuracy and shortcut rate, the fraction of OOD examples on which CoT predicts S. Details are in § G.5.

Table 1: Length/depth generalization. Bins are multiples of the split thresholds $L = 595$ and $K = 10$.

Length	**IID**			**OOD**						
Bin ($\times L$)	≤ 0.25	0.25–0.5	0.5–1.0	1.0–1.25	1.25–1.5	1.5–2.0	2.0–2.5	2.5–3.0	3.0–5.0	> 5.0
RM acc. (%)	100.0	100.0	99.6	99.3	99.3	99.6	98.6	95.7	92.4	76.4
CoT acc. (%)	100.0	100.0	99.6	79.0	24.5	14.2	10.6	6.2	10.1	9.7
Gap (pt)	+0.0	+0.0	+0.0	+20.2	+74.8	+85.4	+88.0	+89.4	+82.4	+66.7
Depth	**IID**			**OOD**						
Bin ($\times K$)	≤ 0.3	0.4–0.6	0.7–1.0	1.1	1.2–1.3	1.4–1.5	1.6–1.7	1.8–1.9	2.0–2.1	≥ 2.2
RM acc. (%)	100.0	99.7	99.6	100.0	99.6	98.6	98.4	99.4	98.6	98.7
CoT acc. (%)	100.0	100.0	99.3	68.6	71.6	67.9	63.1	52.2	48.2	38.4
Gap (pt)	+0.0	−0.3	+0.4	+31.4	+28.0	+30.7	+35.3	+47.2	+50.4	+60.4

Two case studies. In the copy-answer case, S is a deep returned value z that CoT can copy from the trace. In the trace-position case, S is whether the final answer token appears at an even or odd trace position. These are different kinds of shortcuts: one is an internal value token, the other is a global property of the trace. Both are present in the flattened trace and absent from RM's active-frame view. Under OOD evaluation, CoT usually predicts S; RM remains accurate.

Aggregate results. Table 2 reports all six constructions. The remaining rows vary where the shortcut lives: in a hidden intermediate value, in aggregates of internal returns, or in a value-position statistic in the trace. CoT is almost perfect IID in every row, so the OOD collapse is not due to an inability to learn these distributions. Once the shortcut is broken, however, CoT's OOD predictions frequently match S. RM remains much more stable. These errors are not random extrapolation failures; they track trace-level rules that were valid only in training. This supports the shortcut mechanism from § 5: CoT often follows trace-level cues hidden from RM. The strict description-length gap in Theorem 6 remains a condition of the theoretical result.

Table 2: Shortcut behavior across six constructions.

Construction	**IID**		**OOD**			
	RM acc.	**CoT acc.**	**RM acc.**	**CoT acc.**	**Gap (pt)**	**Shortcut rate**
Copy answer from deep return (Case 1)	100.0%	100.0%	87.3%	7.0%	+80.3	92.7%
Hidden intermediate literal	90.3%	99.7%	86.0%	0.3%	+85.7	99.3%
Long-range aggregate	99.7%	100.0%	95.7%	10.3%	+85.3	88.0%
Near-answer aggregate	100.0%	100.0%	99.7%	1.0%	+98.7	79.7%
Position-weighted aggregate	100.0%	100.0%	97.0%	14.7%	+82.3	69.7%
Trace-position parity (Case 2)	100.0%	100.0%	99.7%	9.7%	+90.0	90.3%

Relation to the theory. The experiments compare two training views of the same executions: flattened traces for CoT and active frames for RM. Visibility, sequence segmentation, and positional exposure therefore vary together in this comparison. The executor also permits calls inside return payloads (§ G.1); the simulation theorem covers the restricted protocol with control-free call and return payloads.

7 Conclusion

Recursive language models can be more sample-efficient in our IID experiments, while CoT catches up with enough data. The simulation theorem gives comparable description-length upper bounds for the two learners. The separation appears out of domain because the broader CoT view can make a wrong rule easier to describe than the intended subproblem rule. Our bitter lesson for out-of-domain reasoning is that coverage can hurt: seeing more can make the wrong rule easier to learn. Reasoning beyond the training domain requires more than containing the right rule; it requires the specificity that makes the learner use it.

AI USE STATEMENT

Generative AI tools were used to assist with writing and polishing text, drafting sections of the paper, and developing proofs of mathematical claims. The authors take responsibility for the content of this paper, including its mathematical claims, proofs, and references.

REPRODUCIBILITY STATEMENT

The learning setup and assumptions are specified in § 3. Proofs of the simulation and IID generalization results are provided in §§ D and E, and experimental details, including data generation, model architectures, training, and evaluation, are given in § G.

REFERENCES

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Peter L Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear VC-dimension bounds for piecewise polynomial networks. *Neural Computation*, 10(8):2159–2173, 1998.

Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Peter D Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, 2017.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*, 2023.

Soochan Lee and Gunhee Kim. Recursion of thought: A divide-and-conquer approach to multi-context reasoning with language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 623–658. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.40. URL https://aclanthology.org/2023.findings-acl.40/.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

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.

Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. In *International Conference on Learning Representations*, 2021.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations (Workshop)*, 2015.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

OpenAI. Orchestration and handoffs. `https://developers.openai.com/api/docs/guides/agents/orchestration`, 2026. Accessed July 22, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2023. URL `https://arxiv.org/abs/2310.08560`.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. In *Conference on Language Modeling*, 2025. URL `https://openreview.net/forum?id=YgwQ7sXPXU`.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 78(5):947–1012, 2016.

Mohammad Pezeshki, Sékou-Oumar Kaba, Yoshua Bengio, Aaron Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 1256–1272, 2021.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024.

Prithvi Rajasekaran, Ethan Dixon, Carly Ryan, and Jeremy Hadfield. Effective context engineering for AI agents. `https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents`, 2025. Anthropic Engineering Blog.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In *International Conference on Machine Learning*, pp. 8346–8356, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Philip Schroeder, Nathaniel W. Morgan, Hongyin Luo, and James R. Glass. THREAD: Thinking deeper with recursive spawning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8418–8442, 2025.

Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9573–9585, 2020.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(70):1–57, 2018.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon LLM agent via Context-Folding, 2025. URL https://arxiv.org/abs/2510.11967.

Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization, 2025. URL https://arxiv.org/abs/2509.13313.

Ruize Xu, Chenxiao Yang, Yanhong Li, and David McAllester. Training transformers as a universal computer. *arXiv preprint arXiv:2604.25166*, 2026.

Chenxiao Yang, Nathan Srebro, David McAllester, and Zhiyuan Li. PENCIL: Long thoughts with short memory. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 71092–71156. PMLR, 2025. URL https://proceedings.mlr.press/v267/yang25ac.html.

Chenxiao Yang, Nathan Srebro, and Zhiyuan Li. Recursive models for long-horizon reasoning. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.02112.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models, 2025a. URL https://arxiv.org/abs/2512.24601.

Zhenyu Zhang, Tianyi Chen, Weiran Xu, Alex Pentland, and Jiaxin Pei. ReCAP: Recursive context-aware reasoning and planning for large language model agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 95158–95186, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/88af3540325dd0b70617a9ab605f294d-Abstract-Conference.html.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023.

A Related Work

Our work is closest to recursive and context-isolated approaches to language-model reasoning. Zhang et al. (2025a) introduce *recursive language models*, where an LLM treats a long prompt as an external environment and recursively inspects or decomposes selected snippets; related systems use recursive spawning, divide-and-conquer reasoning, context folding, adaptive decomposition, or planner-executor structure to extend long-context reasoning and planning (Yang et al., 2026; Lee & Kim, 2023; Prasad et al., 2024; Schroeder et al., 2025; Pan et al., 2025; Sun et al., 2025; Zhang et al., 2025b). These works motivate the same design principle that the model should sometimes see a fresh subproblem sequence rather than the entire complete sequence. They mainly ask how recursion or context management improves inference, computation, or task accuracy. We ask a different statistical question: after setting aside the context-length advantage, does context isolation change what is learned? This places the paper between several lines of work. Chain-of-thought, scratchpads, least-to-most prompting, decomposed prompting, and tree-of-thought expose or organize intermediate computation in a visible sequence (Nye et al., 2021; Wei et al., 2022; Zhou et al., 2023; Khot et al., 2023; Yao et al., 2023a). Retrieval, memory, summarization, and agent/tool systems choose which information is shown to the model at each step (Yang et al., 2025; Packer et al., 2023; Wang et al., 2023; Wu et al., 2025; Yu et al., 2025; Yao et al., 2023b; Schick et al., 2023; Hong et al., 2024; Wu et al., 2023). Classical Occam/MDL arguments and the bitter-lesson view explain why a broader CoT learner need not pay much for IID prediction: a general function approximator can still contain the right recursive rule compactly (Rissanen, 1978; Grünwald, 2007; Sutton, 2019). Shortcut-learning, simplicity-bias, and spurious-correlation work point in the opposite direction under distribution shift: information that is harmless in distribution can support easier or shorter non-invariant explanations that fail off domain (Geirhos et al., 2020; Shah et al., 2020; Pezeshki et al., 2021; Sagawa et al., 2020; Nagarajan et al., 2021; Shi et al., 2023; Liu et al., 2024; Peters et al., 2016; Arjovsky et al., 2019). Our contribution is to put these views together. CoT can simulate recursive computation with comparable description length, yielding comparable IID sample-complexity upper bounds, but the complete sequence also exposes shorter non-invariant explanations; recursive context isolation helps OOD generalization by removing those explanations from the learner's view.

B Background: Prefix Codes and the Realizable MDL Bound

This appendix provides the technical background for the MDL framework used in § 2. § B.1 explains how prefix-code lengths can be read as probability weights, which is the formal basis for interpreting a complexity measure as a prior. § B.2 gives a self-contained proof of Theorem 9.

B.1 Prefix Codes and the Code–Distribution Correspondence

Let $\mathcal{U}$ be a countable set. A *code* for $\mathcal{U}$ is a mapping $C : \mathcal{U} \to \{0,1\}^*$ that assigns each $u \in \mathcal{U}$ a binary codeword $C(u)$; its *codelength function* is $L_C(u) = |C(u)|$. A code is *prefix-free* if no codeword is a prefix of any other, so that a concatenation $C(u_1)C(u_2)\cdots C(u_n)$ is uniquely decodable from left to right (Cover & Thomas, 2006). The Kraft inequality characterizes which codelength assignments are realizable by prefix-free codes.

Theorem 7 (Kraft Inequality)**.** *For nonnegative integer lengths $L(u_1), L(u_2), \ldots$, a prefix-free code over $\mathcal{U}$ with these codelengths exists if and only if*

$$\sum_{u \in \mathcal{U}} 2^{-L(u)} \leq 1. \tag{11}$$

Code lengths as probability weights. The Kraft inequality lets prefix-code lengths be read as (sub-)probability weights. Given a prefix code C with lengths $L_C(u)$,

$$P(u) = 2^{-L_C(u)} \tag{12}$$

is a (sub-)probability distribution on $\mathcal{U}$: short codewords correspond to high probability. Conversely, any distribution P on $\mathcal{U}$ gives valid prefix-code lengths

$$L_C(u) = \lceil -\log_2 P(u) \rceil, \tag{13}$$

since $\sum_u 2^{-\lceil -\log_2 P(u) \rceil} \leq \sum_u P(u) = 1$. The ceiling contributes at most one bit of overhead.

Proposition 8 (Code Lengths and Distributions)**.** *Prefix-code lengths over $\mathcal{U}$ and (sub-)probability weights on $\mathcal{U}$ are interchangeable up to integer rounding (Cover & Thomas, 2006, Theorems 5.2.1, 5.3.1): lengths $L(u)$ satisfying Kraft define weights $2^{-L(u)}$, and any distribution P gives valid lengths $\lceil -\log_2 P(u) \rceil$. Assigning probability $P(u)$ to an outcome is therefore equivalent to encoding it in $-\log_2 P(u)$ bits, up to rounding.*

In particular, any complexity measure ℓ on predictors induces a (sub-)prior $2^{-\ell(h)}$, under which shorter descriptions receive higher prior weight. This is the reading of ℓ as a prior used in the main text.

B.2 Proof of the Realizable MDL Bound

We state the standard realizable MDL bound and give a self-contained proof following the Occam argument (Shalev-Shwartz & Ben-David, 2014, Section 7.3).

Theorem 9 (Realizable MDL Generalization Bound)**.** *Fix a complexity measure ℓ, a target f^* with $|f^*|_\ell < +\infty$, and $\delta \in (0,1)$. Let $\mathcal{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ be n i.i.d. samples from $\mathcal{D}$ on Ω, and write $\mathcal{L}_\mathcal{D}(h) = \Pr_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f^*(\mathbf{x})]$. For $\hat{f} = \mathsf{MDL}_\ell(\mathcal{X}, f^*)$, with probability at least $1-\delta$,*

$$\mathcal{L}_\mathcal{D}(\hat{f}) \leq \frac{|\hat{f}|_\ell \ln 2 + \ln(1/\delta)}{n} \leq \frac{|f^*|_\ell \ln 2 + \ln(1/\delta)}{n}. \tag{14}$$

The proof uses the realizable Occam argument: a predictor with large population error is unlikely to fit all training labels, and the Kraft inequality lets us union bound over predictors.

Step 1: A single consistent predictor. Write $\widehat{\mathcal{L}}(h) = \frac{1}{n}\sum_{i=1}^n \mathbb{1}(h(\mathbf{x}^{(i)}) \neq f^*(\mathbf{x}^{(i)}))$. Fix any predictor h, and write $p = \mathcal{L}_\mathcal{D}(h)$. The probability that h agrees with f^* on all n samples is

$$\Pr[\widehat{\mathcal{L}}(h) = 0] = (1-p)^n \leq \exp(-np). \tag{15}$$

Therefore, for any threshold $\epsilon > 0$,

$$\Pr\Big[\widehat{\mathcal{L}}(h) = 0 \text{ and } \mathcal{L}_\mathcal{D}(h) > \epsilon\Big] \leq \exp(-n\epsilon). \tag{16}$$

Step 2: Weighted union bound via the Kraft inequality. Set

$$\epsilon_h = \frac{|h|_\ell \ln 2 + \ln(1/\delta)}{n}. \tag{17}$$

By Equation (15),

$$\Pr\Big[\widehat{\mathcal{L}}(h) = 0 \text{ and } \mathcal{L}_\mathcal{D}(h) > \epsilon_h\Big] \leq \exp(-n\epsilon_h) = \delta\, 2^{-|h|_\ell}. \tag{18}$$

Taking a union bound over all finite-length predictors h and using Kraft,

$$\sum_h \delta\, 2^{-|h|_\ell} \leq \delta. \tag{19}$$

Thus, with probability at least $1-\delta$, no predictor is both consistent with the sample and has population loss above its threshold ϵ_h. Equivalently, every h with $\widehat{\mathcal{L}}(h) = 0$ satisfies

$$\mathcal{L}_\mathcal{D}(h) \leq \frac{|h|_\ell \ln 2 + \ln(1/\delta)}{n}. \tag{20}$$

MDL corollary. The MDL learner $\hat{f} = \mathsf{MDL}_\ell(\mathcal{X}, f^*)$ agrees with f^* on every input in $\mathcal{X}$, so $\widehat{\mathcal{L}}(\hat{f}) = 0$. Moreover, by definition $|\hat{f}|_\ell \leq |f^*|_\ell$. Substituting:

$$\mathcal{L}_\mathcal{D}(\hat{f}) \leq \frac{|\hat{f}|_\ell \ln 2 + \ln(1/\delta)}{n} \leq \frac{|f^*|_\ell \ln 2 + \ln(1/\delta)}{n}. \tag{21}$$

This completes the proof. □

B.3 Examples of Complexity Measures

The simplest version of this argument is the finite-class bound. A standard IID guarantee for a finite class $\mathcal{F}$ depends on the size of the class only through $\log|\mathcal{F}|$, up to accuracy and confidence factors. Thus if a generic class $\mathcal{F}_{\text{gen}}$ contains an expert class $\mathcal{F}_{\text{expert}}$ and $|\mathcal{F}_{\text{gen}}| = 10^6|\mathcal{F}_{\text{expert}}|$, then replacing the expert class by the generic one adds only $\log_2 10^6 \approx 20$ to the usual log-class-size term. A broad class can contain many irrelevant predictors without making IID learning impossible, as long as it covers a good rule.

The term *description length* is justified by the Kraft condition in the definition: it is the standard prefix-code condition that lets $|f|_\ell$ be read as the number of bits needed to describe f. Thus a complexity measure can also be viewed as a *description language*; we use the two terms interchangeably. The key consequence is scarcity: at most 2^k predictors can have description length at most k (see § B for the code interpretation). Thus ℓ encodes a built-in preference: before seeing the data, shorter descriptions are treated as more plausible (Shalev-Shwartz & Ben-David, 2014).

This definition includes several standard complexity measures. A traditional hypothesis class is recovered by assigning finite length to predictors in the class and length $+\infty$ to predictors outside it. For a finite class $\mathcal{F}$, the uniform choice gives length $\log_2|\mathcal{F}|$ to each predictor in $\mathcal{F}$. Parameter count gives another example: if a model f_θ is stored with b bits per parameter, then its description length is proportional to $b|\theta|$. Norm-based measures are also common proxies for simplicity, motivated by explicit regularization and by the implicit bias of gradient descent toward lower-complexity interpolating solutions (Neyshabur et al., 2015; Gunasekar et al., 2017; Soudry et al., 2018). Formally, continuous parameter classes require a discretization or coding scheme; throughout, our bounds apply to the resulting countable set of predictors.

B.4 Coverage over Specificity in Distribution

Think of ℓ_A as a specialized description language and ℓ_B as a more general one. The general language may describe many additional predictors, including irrelevant ones. For IID prediction, this extra coverage is harmless if every rule that was concise under ℓ_A is still concise under ℓ_B.

We say that ℓ_B *dominates* ℓ_A with constant factor $C \geq 1$ (written $\ell_A \preceq \ell_B$) if

$$|f|_{\ell_B} \leq C|f|_{\ell_A} \tag{22}$$

for every predictor f with finite ℓ_A-length. This is the description-length version of moving from an expert class to a generic function approximator: the generic language may express more functions, but it does not make the expert solutions much longer to describe. For parameter-count codes, this condition holds when every specialized predictor can be implemented in the general architecture with at most a constant-factor increase in parameters.

Corollary 10 (Expressiveness Is Cheap for IID Prediction)**.** *If $\ell_A \preceq \ell_B$ and $\hat{f}_A = \mathsf{MDL}_{\ell_A}(\mathcal{X}, f^*)$, $\hat{f}_B = \mathsf{MDL}_{\ell_B}(\mathcal{X}, f^*)$, then*

$$|\hat{f}_B|_{\ell_B} \leq C|\hat{f}_A|_{\ell_A}. \tag{23}$$

Hence the IID upper bound under the broader language changes only by a constant factor in the description-length term, and so does the corresponding sample-complexity upper bound, up to confidence terms.

Indeed, $\hat{f}_A$ fits the same training data, so it is a feasible candidate for the ℓ_B-MDL problem. Domination gives $|\hat{f}_A|_{\ell_B} \leq C|\hat{f}_A|_{\ell_A}$, and the ℓ_B-MDL output can only be no longer than this candidate.

Sweet lesson. This is the sweet side of Sutton's bitter lesson (Sutton, 2019): for IID prediction, coverage is more important than perfect specificity. A more general language is not penalized merely because it can express more functions; the IID guarantee pays for the compact rule that learning returns.

There is one important limitation. The corollary compares risks on the training distribution; it does not say that MDL selects the same rule under the two languages. Two predictors can both fit the training data and receive comparable IID guarantees, while disagreeing elsewhere. That difference is invisible to the IID bound, but it is exactly what matters once we ask whether the learned rule works beyond the training distribution.

B.5 KRAFT VERIFICATION FOR THE LIFTED MEASURE

The lifted measure in Definition 4 is again a valid complexity measure: choosing one shortest local representative for each finite-length f gives distinct representatives, so the Kraft sum for ℓ^{Π} is bounded by the Kraft sum for ℓ. The local-fitting equivalence is stated in Proposition 1.

Under ℓ^{Π}, finite-length complete-sequence rules are exactly those whose predictions have the form $g(\Pi(\mathbf{x}))$ for some local rule g. Rules that cannot be written this way have length $+\infty$. For the remaining rules, the cost is the shortest local rule before composing with Π. Thus lifted MDL minimizes $|g|_{\ell}$; this is different from minimizing the original length $|g \circ \Pi|_{\ell}$ of the composed complete-sequence rule.

C TRANSFORMER ARCHITECTURE

This appendix provides a self-contained formal definition of the decoder-only Transformer architecture used throughout the paper. Our formulation follows the standard architecture of Vaswani et al. (2017), with hard attention as the default attention mechanism, following recent theoretical work (Hahn, 2020; Merrill et al., 2022). We assume exact arithmetic over the reals unless stated otherwise.

Let Σ be a finite vocabulary with $|\Sigma| = K$, let $\Omega := \Sigma^*$, and write $\|$ for concatenation. Token sequences are written as $\mathbf{x} = (x_1, x_2, \ldots, x_T) \in \Sigma^T$, where T is the sequence length. Embedding sequences are written as $H \in \mathbb{R}^{T \times d}$, where d is the embedding dimension. We use $H_i \in \mathbb{R}^d$ to denote the embedding at position i.

Transformer layers. A Transformer of depth L transforms an input embedding sequence $H^{(0)} = H \in \mathbb{R}^{T \times d}$ through L layers. We allow layer-dependent hidden dimensions $d^{(0)}, d^{(1)}, \ldots, d^{(L)}$ with $d^{(0)} = d$, so $H^{(\ell)} \in \mathbb{R}^{T \times d^{(\ell)}}$. Each layer applies two sublayers: an attention mechanism followed by a feed-forward network:

$$\tilde{H}^{(\ell)} = \mathsf{Attn}^{(\ell)}(H^{(\ell-1)}), \qquad H^{(\ell)} = \mathsf{FFN}^{(\ell)}(\tilde{H}^{(\ell)}). \tag{24}$$

Our results hold regardless of whether residual connections are included.

Hard attention. The attention mechanism enables data-dependent selection of information across positions. For each attention head, queries, keys, and values are computed via affine transformations:

$$Q_i = W_Q H_i + b_Q, \qquad K_i = W_K H_i + b_K, \qquad V_i = W_V H_i + b_V, \tag{25}$$

where W_Q, W_K, W_V are projection matrices and b_Q, b_K, b_V are bias vectors (suppressing the layer index ℓ). Each head may also include an output affine map W_O, b_O. We use *causal masking*: position i can only attend to positions $j \le i$. We analyze *hard attention* (Hahn, 2020; Merrill et al., 2022), where attention selects a score-maximizing position. Let

$$M_i = \left\{ j \le i : \langle Q_i, K_j \rangle = \max_{k \le i} \langle Q_i, K_k \rangle \right\} \tag{26}$$

denote the set of causally-visible positions achieving the maximum score. We use rightmost-hard attention: let $j_i^* := \max M_i$ and define $\alpha_{i,j} = \mathbf{1}[j = j_i^*]$. The pre-output attention value is $U_i = \sum_j \alpha_{i,j} V_j$, and the head output is $W_O U_i + b_O$ when an output map is present. The standard MDL bound in Theorem 9 does not depend on this convention. The comparison in Theorem 4 uses the simulation construction of Theorem 2 and its rightmost-hard convention (Hahn, 2020).

Multi-head attention. A multi-head attention layer at layer ℓ with $h^{(\ell)}$ heads computes $h^{(\ell)}$ attention outputs in parallel. Each head $r \in \{1, \ldots, h^{(\ell)}\}$ has its own learned block of affine maps $W_Q^{(\ell,r)}, W_K^{(\ell,r)}, W_V^{(\ell,r)}, W_O^{(\ell,r)}$ and the corresponding biases, and applies hard attention as defined above to produce $\mathsf{Attn}^{(\ell,r)}(H)$. The outputs are summed:

$$\mathsf{Attn}^{(\ell)}(H) = \sum_{r=1}^{h^{(\ell)}} \mathsf{Attn}^{(\ell,r)}(H). \tag{27}$$

In practice, Transformers partition the input dimension across heads, concatenate their outputs, and apply a linear projection; the summation formulation is an equivalent notational simplification for the arguments below.

Feed-forward network. The FFN sublayer at layer ℓ is a position-wise MLP with ReLU activations, mapping $\mathbb{R}^{d^{(\ell-1)}} \to \mathbb{R}^{d^{(\ell)}}$. It has internal depth $D^{(\ell)}$ and hidden dimensions specified by a sequence $\bar{d}^{(\ell)} \in \mathbb{N}^{D^{(\ell)}+1}$ where the first element equals $d^{(\ell-1)}$ and the last equals $d^{(\ell)}$. The results extend to any piecewise-linear activation (Bartlett et al., 1998).

Product gates (for the simulation theorem). The simulation construction of Theorem 2 (§ D) additionally assumes the FFN class admits coordinatewise product gates, equivalently ReGLU-style gates: a constant-depth position-wise network can compute a bilinear form $\langle u, v \rangle$ between two hidden vectors $u, v \in \mathbb{R}^{d^{(\ell)}}$. The standard MDL bound in Theorem 9 applies independently of this architectural choice; the CoT–RM comparison in Theorem 4 uses the simulation theorem and inherits its assumptions.

Input layer. The Transformer operates on token sequences $\mathbf{x} \in \Sigma^T$. The *token embedding* $\mathsf{TE} : \Sigma \to \mathbb{R}^d$, parametrized by $W_{\mathsf{TE}} \in \mathbb{R}^{d \times K}$, maps each token to a d-dimensional vector. The *positional embedding* $\mathsf{PE} : \{1, \ldots, T\} \to \mathbb{R}^d$ encodes position information. The input embedding sequence is:

$$H = \big(\mathsf{TE}(x_1) + \mathsf{PE}(1),\ \ldots,\ \mathsf{TE}(x_T) + \mathsf{PE}(T)\big) \in \mathbb{R}^{T \times d}. \tag{28}$$

Our architecture allows any fixed (non-learnable) positional embedding PE.

Output layer. The output is extracted from the final position $H_T^{(L)} \in \mathbb{R}^{d^{(L)}}$ of the last layer. A decoding matrix $W_{\mathsf{DE}} \in \mathbb{R}^{K \times d^{(L)}}$ maps the final representation to logits over the vocabulary:

$$f_\theta(\mathbf{x}) = \arg\max_{a \in \Sigma} \big[W_{\mathsf{DE}}\, H_T^{(L)}\big]_a \in \Sigma. \tag{29}$$

In practice, W_{DE} may share weights with the token embedding W_{TE} (*weight tying*); the analysis permits both tied and untied configurations.

Hypothesis classes. A Transformer architecture $\mathcal{A}$ is fully specified by:

$$\mathcal{A} = \big\{K,\ L,\ \{d^{(\ell)}\}_{\ell=0}^{L},\ \{h^{(\ell)}\}_{\ell=1}^{L},\ \{\bar{d}^{(\ell)}\}_{\ell=1}^{L},\ \mathsf{PE}\big\}, \tag{30}$$

encoding vocabulary size K, depth L, layer-wise hidden dimensions, attention heads per layer, FFN internal dimensions, and positional encoding. We write $L(\mathcal{A}) := L$ for the depth and $W(\mathcal{A})$ for the number of independently stored learned parameters. Standard reuse of the same layer across sequence positions and autoregressive time steps is counted once. In lifted architectures, lane-specific or subroutine-specific copies of a learned block are counted separately, while fixed wiring and fixed non-learned operations are not counted. Equivalently, in the main text we write $L_{\mathcal{A}} := L(\mathcal{A})$ and $W_{\mathcal{A}} := W(\mathcal{A})$. For the lifted constructions below, the architecture specification may also include several named hidden-state lanes and fixed non-learned operations, as described in § C.1. For a fixed architecture $\mathcal{A}$, the hypothesis class of next-token generators is

$$\mathcal{F}_{\mathcal{A}} := \{f_\theta : \theta \in \mathbb{R}^{W(\mathcal{A})}\}. \tag{31}$$

We say that a function $f : \Sigma^* \to \Sigma$ is *realized* by architecture $\mathcal{A}$ if $f \in \mathcal{F}_{\mathcal{A}}$, i.e., there exist parameters $\theta \in \mathbb{R}^{W(\mathcal{A})}$ such that $f_\theta(\mathbf{x}) = f(\mathbf{x})$ for all $\mathbf{x} \in \Sigma^*$.

C.1 Lanes and Unfolded Parameter Count

Learned blocks and parameter count. A Transformer architecture specifies a finite list of learned scalar parameters

$$\theta = (\theta_1, \ldots, \theta_{W(\mathcal{A})}) \in \mathbb{R}^{W(\mathcal{A})}. \tag{32}$$

All learned matrices and biases appearing in the computation graph are assembled from this list. We count $W(\mathcal{A})$ as the number of independent learned scalars in this list. Fixed positional features, fixed masks, fixed exact-match selectors, fixed copy operations, fixed merge gates, and other non-learned arithmetic operations are part of the architecture specification and are not counted in $W(\mathcal{A})$.

Lanes. A lane is just a named copy of the hidden-state stream, used to store a different simulated view of the same input sequence. A layer may maintain several such lanes

$$H^{(\ell,r)} = (H_1^{(\ell,r)}, \ldots, H_T^{(\ell,r)}), \qquad r \in \mathcal{R}_\ell, \tag{33}$$

where each lane has a prescribed width.

Unfolded learned-parameter count. In lifted architectures we use the dense unfolded convention. If a construction applies an original learned block to several lanes or subroutines, these applications are represented by independent learned copies and all copies are counted in $W(\mathcal{A})$. In a simulation proof, we may choose these copied parameters to have the same numerical values as the original block. This equality is part of the witness parameter setting, not an architectural weight-sharing constraint.

Fixed positional features, fixed masks, fixed exact-match selectors, fixed copy operations, fixed merge gates, and other non-learned arithmetic operations are part of the architecture specification and are not counted in $W(\mathcal{A})$.

Autoregressive generation. The Transformer serves as a next-token generator $f_\theta : \Omega \to \Sigma$ (§ 3). Given an initial input sequence $\mathbf{x}_0 \in \Omega$, autoregressive generation repeatedly applies f_θ and appends the output:

$$\mathbf{x}_{t+1} = \mathbf{x}_t \parallel f_\theta(\mathbf{x}_t), \qquad t \geq 0, \tag{34}$$

as in § 3. At each step, the same parameters θ are used. In the CoT setting, f_θ reads the complete sequence $\mathbf{x}_t$. In the recursive model setting (§ 3), the predictor reads the active frame $\Pi_{\mathrm{rec}}(\mathbf{x}_t)$.

Teacher-forced training. In practice, autoregressive models are trained using teacher forcing (Williams & Zipser, 1989): at each step, the model predicts the next token conditioned on the ground-truth prefix rather than its own previous predictions. Given a dataset S of prefix–next-token pairs $(\mathbf{x}, y)$ drawn from correct generated sequences, the training objective is the empirical cross-entropy loss:

$$\widehat{\mathcal{L}}(\theta;\, S) = \frac{1}{|S|} \sum_{(\mathbf{x},\, y) \in S} -\log\, p_\theta(y \mid \mathbf{x}), \tag{35}$$

where $p_\theta(a \mid \mathbf{x}) = \mathrm{softmax}(W_{\mathsf{DE}}\, H_T^{(L)})_a$ is the predicted probability of token a given prefix $\mathbf{x}$.

D Proof of Theorem 2 (CoT Simulation)

D.1 Active-Frame Replay Algorithm

The following procedure recovers the active frame from the complete sequence in § 3.

Active-Frame Map

Require: Complete sequence $\mathbf{x} = (x_1, \ldots, x_n) \in \Sigma^*$
$\mathbf{S} \leftarrow [\epsilon]$
for $i = 1, \ldots, n$ **do**
  $\mathbf{S}[-1] \leftarrow \mathbf{S}[-1] \parallel x_i$
  if $\mathbf{S}[-1]$ ends with $\langle \texttt{call} \rangle\, \mathbf{q}\, \langle /\texttt{call} \rangle$ for some $\mathbf{q} \in \Sigma^*_{\mathrm{ord}}$ **then**
    Remove $\langle \texttt{call} \rangle\, \mathbf{q}\, \langle /\texttt{call} \rangle$ from $\mathbf{S}[-1]$
    $\mathbf{S} \leftarrow \mathbf{S} \parallel [\mathbf{q}]$
  else if $\mathbf{S}[-1]$ ends with $\langle \texttt{return} \rangle\, \mathbf{a}\, \langle /\texttt{return} \rangle$ for some $\mathbf{a} \in \Sigma^*_{\mathrm{ord}}$, and $|\mathbf{S}| \geq 2$ **then**
    Pop $\mathbf{S}[-1]$
    $\mathbf{S}[-1] \leftarrow \mathbf{S}[-1] \parallel \mathbf{a}$
  end if
end for
return $\mathbf{S}[-1]$

Proof idea. When we replay the recursive stack update, an ordinary token inside an unfinished call/return block has two possible roles. Before the block closes, it is part of the current unfinished suffix. After the block closes, the same token belongs to the frame that eventually owns it. We therefore maintain two simulated hidden states at every position: an *owner state* H^O for the eventual

frame, and a *pending state* H^P for the unfinished suffix. The proof has three steps. First, we characterize $\Pi_{\mathrm{rec}}(\mathbf{x}_{1:i})$ as either an owner sequence O_j or a pending sequence P_i (§ D.2). Second, we run each layer of $\mathcal{A}$ on both states in parallel (§ D.3). Third, we read out the state corresponding to the current active frame and count resources (§ D.4).

The lifted invariant. Everything below is aimed at maintaining, through every layer $\ell \in \{0, \ldots, L\}$, the invariant

$$\begin{aligned} H_j^{O,(\ell)} &= H_{|O_j|}^{\mathcal{A},\ell}(O_j) \quad \text{for every } x_j \in \Sigma_{\mathrm{ord}}, \\ H_j^{P,(\ell)} &= H_{|P_j|}^{\mathcal{A},\ell}(P_j) \quad \text{for every } j \in \mathcal{P}. \end{aligned} \tag{36}$$

where O_j, P_j are the two imagined input sequences constructed in § D.2 and $\mathcal{P}$ is the set of pending positions.

Assumptions. We work on legal prefixes (Definition 7); non-legal inputs are mapped arbitrarily by $\mathcal{A}'$. The lifted family is rightmost-hard with product-gated FFNs and reindexable positional features (§ C). Here rightmost-hard only fixes the tie-breaking rule, and reindexable positional features mean that the construction can use the positional feature for a simulated position such as $|O_j|$ or $|P_j|$. Without loss of generality every input $\mathbf{x}$ begins with a distinguished sentinel $\langle \mathtt{bos} \rangle \in \Sigma_{\mathrm{ord}}$, and we adopt the convention $g(\epsilon) := g(\langle \mathtt{bos} \rangle)$ for edge cases in which the observation is empty. We use the unfolded parameter-count convention of § C.1: lane-specific and subroutine-specific applications of an original learned block are implemented as independent learned copies in $\mathcal{A}'$, and the simulator parameter setting assigns those copies the same numerical values as the corresponding block of $\mathcal{A}$. For constant-depth claims we use known constant-depth constructions from Yang et al. (2025) (prefix sums, conditional maxima, rightmost exact-match copying, local piecewise-linear computations, $O(\log n)$-precision integer multiplication); together with product gates these suffice to compute bilinear forms over $O(d^{(\ell)})$-dimensional hidden vectors.

Restricted-attention scope. The finite-penalty implementation of restricted attention below is stated for the bounded-precision parameter classes used in § 4; with exact non-learned hard masks for the computed allowed-position sets, the same simulation applies without this bounded-score step.

Proof convention. The construction uses the unfolded convention of § C.1. Each lane-specific or subroutine-specific application of an original learned attention head, FFN, or decoder block is an independent learned copy in $\mathcal{A}'$. To prove simulation, we exhibit a parameter setting in which each copy has the same numerical value as the corresponding block of $\mathcal{A}$. For readability, the formulas below write these copied matrices using the original symbols; this is a notational shorthand and not an architectural weight-sharing assumption. The bookkeeping block, allowed-position sets, exact-match selectors, copy operations, and merge gates remain fixed non-learned parts of the architecture.

D.2 What $\Pi_{\mathrm{rec}}(\mathbf{x})$ Looks Like

The stack replay can view the same ordinary token in two ways. Before the surrounding block closes, the token contributes to the current unfinished suffix; after the close, it contributes to its owner frame. We encode these two views by two imagined input sequences O_j and P_j. These are not extra generated sequences; they are the sequences that the original architecture $\mathcal{A}$ would see if it were run on the relevant frame.

Recall from the recursive-model setup in § 3 that $\Sigma_{\mathrm{ctrl}} = \{\langle \mathtt{call} \rangle, \langle /\mathtt{call} \rangle, \langle \mathtt{return} \rangle, \langle /\mathtt{return} \rangle\}$ and $\Sigma_{\mathrm{ord}} = \Sigma \setminus \Sigma_{\mathrm{ctrl}}$. The replay fires on completed suffix blocks: $\langle /\mathtt{call} \rangle$ triggers a push when the top frame ends with $\langle \mathtt{call} \rangle\, \mathbf{q}\, \langle /\mathtt{call} \rangle$ ($\mathbf{q} \in \Sigma_{\mathrm{ord}}^*$), and $\langle /\mathtt{return} \rangle$ triggers a pop when it ends with $\langle \mathtt{return} \rangle\, \mathbf{a}\, \langle /\mathtt{return} \rangle$ ($\mathbf{a} \in \Sigma_{\mathrm{ord}}^*$).

We only need to consider flattened sequences that the recursive stack procedure could actually produce. The following condition makes that precise.

Definition 7 (Legal prefix)**.** *A prefix $\mathbf{x}_{1:i} \in \Sigma^*$ is* legal *if, during the replay of the procedure in § D.1: (i) every $\langle /\mathtt{call} \rangle$ closes a matching $\langle \mathtt{call} \rangle\, \mathbf{q}\, \langle /\mathtt{call} \rangle$ block with $\mathbf{q} \in \Sigma_{\mathrm{ord}}^*$ and fires the push rule; (ii) every $\langle /\mathtt{return} \rangle$ closes a matching $\langle \mathtt{return} \rangle\, \mathbf{a}\, \langle /\mathtt{return} \rangle$ block with $\mathbf{a} \in \Sigma_{\mathrm{ord}}^*$ and fires the pop rule; (iii) the replay never pops below the root. Equivalently, $\mathbf{x}_{1:i}$ is a prefix of a well-formed recursive generated sequence with control-free payloads.*

Bookkeeping labels. The following labels only keep track of which frame each token belongs to and whether the current top frame has an unfinished call or return block. For each position i, let

$$\begin{aligned}\lambda_i &:= \max\big(\{t < i : x_t \in \Sigma_{\mathrm{ctrl}}\} \cup \{0\}\big),\\ \beta_i &:= \max\big(\{t \le i : x_t \in \Sigma_{\mathrm{ctrl}}\} \cup \{0\}\big),\end{aligned} \tag{37}$$

and the completed depth $c_i := \sum_{t\le i}(\mathbf{1}[x_t = \langle/\texttt{call}\rangle] - \mathbf{1}[x_t = \langle/\texttt{return}\rangle])$. The current committed frame id and its parent are

$$\begin{aligned}F_i &:= \max(\{\lambda_t : t \le i,\ x_t = \langle/\texttt{call}\rangle,\ c_t = c_i\} \cup \{0\}),\\ \mathrm{par}_i &:= \max(\{\lambda_t : t \le i,\ x_t = \langle/\texttt{call}\rangle,\ c_t = c_i - 1\} \cup \{0\}).\end{aligned} \tag{38}$$

The pending-block type is

$$m_i := \begin{cases}\mathsf{call}, & \beta_i > 0 \text{ and } x_{\beta_i} = \langle\texttt{call}\rangle,\\ \mathsf{return}, & \beta_i > 0 \text{ and } x_{\beta_i} = \langle\texttt{return}\rangle,\\ \bot, & \text{otherwise.}\end{cases} \tag{39}$$

Lemma 11 (Replay normal form)**.** *For every legal prefix $\mathbf{x}_{1:i}$:*

1. *Every non-top frame of the stack is a string in Σ^*_{ord}.*
2. *The top frame has exactly one of the three forms $\mathbf{U}$, $\mathbf{U} \,\|\, \langle\texttt{call}\rangle \,\|\, \mathbf{q}$, $\mathbf{U} \,\|\, \langle\texttt{return}\rangle \,\|\, \mathbf{a}$ with $\mathbf{U}, \mathbf{q}, \mathbf{a} \in \Sigma^*_{\mathrm{ord}}$.*
3. *In particular, the top frame contains at most one unfinished block; if one exists it is the suffix starting at x_{β_i}.*
4. *$m_i \neq \bot$ iff $\Pi_{\mathrm{rec}}(\mathbf{x}_{1:i})$ ends with the unfinished suffix $x_{\beta_i} \cdots x_i$.*

Proof. By induction on i. Base case $i = 0$: stack $[\epsilon]$ (form $\mathbf{U} = \epsilon$). Step: appending $x_i \in \Sigma_{\mathrm{ord}}$ extends $\mathbf{U}$, $\mathbf{q}$, or $\mathbf{a}$; appending $\langle\texttt{call}\rangle$ or $\langle\texttt{return}\rangle$ requires the first form (legality forbids nested unfinished blocks) and produces the second or third; $\langle/\texttt{call}\rangle$ and $\langle/\texttt{return}\rangle$ contract the second or third back to the first and push/pop. Clauses 3–4 follow from the three-form classification. □

Owner view. For $x_j \in \Sigma_{\mathrm{ord}}$, define the *owner frame id*

$$\omega_j := \begin{cases}\lambda_j, & \lambda_j > 0 \text{ and } x_{\lambda_j} = \langle\texttt{call}\rangle,\\ \mathrm{par}_{\lambda_j}, & \lambda_j > 0 \text{ and } x_{\lambda_j} = \langle\texttt{return}\rangle,\\ F_j, & \text{otherwise,}\end{cases} \tag{40}$$

and the *owner imagined input*

$$O_j := \big(x_s : 1 \le s \le j,\ x_s \in \Sigma_{\mathrm{ord}},\ \omega_s = \omega_j\big), \tag{41}$$

so that x_j occupies position $|O_j|$ within O_j.

Pending view. Define the *pending set*

$$\mathcal{P} := \{j : x_j \in \{\langle\texttt{call}\rangle, \langle\texttt{return}\rangle\}\} \cup \{j : x_j \in \Sigma_{\mathrm{ord}},\ \lambda_j > 0,\ x_{\lambda_j} \in \{\langle\texttt{call}\rangle, \langle\texttt{return}\rangle\}\}. \tag{42}$$

For $j \in \mathcal{P}$, let $\mathrm{host}_j := F_{\beta_j}$ be the host-frame id, and the *pending imagined input*

$$P_j := \big(x_s : s < \beta_j,\ x_s \in \Sigma_{\mathrm{ord}},\ \omega_s = \mathrm{host}_j\big) \,\|\, x_{\beta_j} \cdots x_j. \tag{43}$$

Lemma 12 (Explicit replay formula)**.** *For every legal prefix $\mathbf{x}_{1:i}$, let $C_i := \{j \le i : x_j \in \Sigma_{\mathrm{ord}},\ \omega_j = F_i\}$. Then*

$$\Pi_{\mathrm{rec}}(\mathbf{x}_{1:i}) = \begin{cases}O_{j_i^*}, & m_i = \bot,\ C_i \neq \varnothing,\ j_i^* := \max C_i,\\ P_i, & m_i \neq \bot,\\ \epsilon, & m_i = \bot,\ C_i = \varnothing.\end{cases} \tag{44}$$

Proof. If $m_i = \bot$: Lemma 11 gives top frame $\mathbf{U}$, i.e., the ordered ordinary tokens of frame F_i, namely C_i; if $C_i \neq \varnothing$, this is $O_{j_i^*}$. If $m_i \neq \bot$: the top frame is the committed host body followed by the unfinished suffix, i.e., P_i. □

D.3 Simulating One Layer of $\mathcal{A}$

We now simulate each layer of $\mathcal{A}$ in parallel on both imagined inputs. The owner state updates at every ordinary token, including call- and return-payload tokens inside an unfinished block; the pending state updates only at positions in the unique unfinished suffix.

Lemma 13 (Bookkeeping labels are computable)**.** *All labels used below—$\lambda, c, F, \mathrm{par}, m, \beta, \omega, \mathrm{host}$, the allowed-position sets $J_i^O, J_i^{\mathrm{host}}, J_i^{\mathrm{blk}}$ (defined below), and the host-empty indicator HE_i—are computable by a constant-depth causal Transformer block using $O(1)$ numeric channels, each storing an $O(\log n)$-bit integer or a rational of polynomial bit complexity (Yang et al., 2025).*

The point of this lemma is only that these quantities are deterministic bookkeeping computed from the control tokens; they are not learned parameters.

Proof. λ_i, β_i select the most recent control-token positions satisfying fixed conditions; c_i is a prefix sum; F_i, par_i select the most recent matching frame ids among $O(\log n)$-bit integers; ω_j is local given $F_j, \lambda_j, \mathrm{par}_{\lambda_j}$; $\mathrm{host}_j = F_{\beta_j}$ is local. The host-empty indicator is the complement of an OR over earlier positions:

$$\mathsf{HE}_i \;=\; 1 - \mathbf{1}\big[\exists s < \beta_i :\; x_s \in \Sigma_{\mathrm{ord}} \;\wedge\; \omega_s = \mathrm{host}_i\big], \tag{45}$$

and the existential is the OR of $\mathbf{1}[s < \beta_i] \cdot \mathbf{1}[x_s \in \Sigma_{\mathrm{ord}}] \cdot \mathbf{1}[\omega_s = \mathrm{host}_i]$ over earlier positions s; the final complement is local. The three allowed-position sets are Boolean combinations of integer equalities and order comparisons among these labels. Each of these operations is known to be implementable in constant depth (Yang et al., 2025). □

Two local subroutines. The lifted layer uses only two simple attention subroutines. First, we restrict a head to the positions that are allowed to appear in the simulated frame. Second, when the simulated frame is a host frame followed by an unfinished suffix, we merge the best position from the host part with the best position from the suffix part.

Restricting attention by penalties. To make a head ignore disallowed positions without changing how it ranks allowed positions, we add a large negative penalty only to the disallowed positions. Concretely, suppose an allowed-position predicate $\mathsf{Allow}(i, j)$ has constant-width query/key features a_i, b_j, computable from the bookkeeping labels, such that

$$\langle a_i, b_j \rangle = 0 \qquad \text{whenever } \mathsf{Allow}(i, j) = 1, \tag{46}$$

$$\langle a_i, b_j \rangle \le -1 \qquad \text{whenever } \mathsf{Allow}(i, j) = 0. \tag{47}$$

Lemma 14 (Restricted rightmost head)**.** *Fix query, key, and value streams Q_i, K_j, V_j, and an allowed-position predicate $\mathsf{Allow}(i, j)$ with penalty features a_i, b_j as above. Suppose every original score satisfies*

$$|\langle Q_i, K_j \rangle| \le \Gamma_{\mathcal{A}}(i). \tag{48}$$

Let $M_{\mathcal{A}}(i) := 2\Gamma_{\mathcal{A}}(i) + 1$. Then rightmost-hard attention with score

$$\widetilde{S}_{ij} := \langle Q_i, K_j \rangle + M_{\mathcal{A}}(i) \langle a_i, b_j \rangle \tag{49}$$

selects exactly the rightmost maximizer of $\langle Q_i, K_j \rangle$ among allowed keys, whenever there is at least one allowed key.

Proof. Allowed keys receive zero shift, so their relative ordering and rightmost tie rule are unchanged. Every disallowed key receives a penalty at most $-M_{\mathcal{A}}(i)$. Since $M_{\mathcal{A}}(i) > 2\Gamma_{\mathcal{A}}(i)$, every allowed key strictly dominates every disallowed key. Therefore the rightmost-hard winner is exactly the rightmost maximizer among allowed keys. □

The allowed-position sets $J_i^O, J_i^{\mathrm{host}}, J_i^{\mathrm{blk}}$ used below have such penalty features. They are Boolean combinations of integer equalities and order comparisons among $\lambda, \beta, \omega, \mathrm{host}, i, j$, all stored in $O(\log n)$-bit channels. Equalities are enforced by quadratic penalties, inequalities by fixed integer-comparison penalties, and conjunctions by summing penalties.

Lemma 15 (Merging the host part and the unfinished suffix). *Let $Y = Y^H \,\|\, Y^K$ with Y^H, Y^K both non-empty and every position of Y^K strictly right of every position of Y^H. Suppose two applications of Lemma 14 return the rightmost-hard (value, key) winners (v^H, k^H) on Y^H and (v^K, k^K) on Y^K. For a query q, set $s^H := \langle q, k^H \rangle$, $s^K := \langle q, k^K \rangle$, and*

$$v := \begin{cases} v^H, & s^H > s^K, \\ v^K, & s^H \le s^K. \end{cases} \tag{50}$$

Then v equals the rightmost-hard output on Y with query q.

Proof. Rightmost-hard on Y selects the rightmost maximizer. Since Y^K is strictly right of Y^H, any cross-tie resolves to the block; the strict inequality $s^H > s^K$ lets host win only when host strictly dominates. □

Fix any architecture $\mathcal{A}$ of depth $L = L(\mathcal{A})$ realizing the local recursive rule g.

Lifted state. At each position j and layer ℓ, $\mathcal{A}'$ maintains

$$\widetilde{H}_j^{(\ell)} = \big[H_j^{O,(\ell)},\ H_j^{P,(\ell)},\ W_j^{(\ell)},\ M_j\big] \in \mathbb{R}^{d^{(\ell)}} \oplus \mathbb{R}^{d^{(\ell)}} \oplus \mathbb{R}^{O(d^{(\ell)})} \oplus \mathbb{R}^{O(1)}, \tag{51}$$

where $W^{(\ell)}$ stores temporary quantities used inside the layer and cleared afterward, and M stores the bookkeeping labels of Lemma 13.

Initialization. $H_j^{O,(0)} := \mathsf{TE}(x_j) + \mathsf{PE}(|O_j|)$ for $x_j \in \Sigma_{\mathrm{ord}}$; $H_j^{P,(0)} := \mathsf{TE}(x_j) + \mathsf{PE}(|P_j|)$ for $j \in \mathcal{P}$; other state entries are zero. Since the positional features can be evaluated at the simulated positions $|O_j|$ and $|P_j|$, these initial states are constant-depth computable from $\mathbf{x}$ and the bookkeeping labels.

Bounding scores before adding penalties. Under the bounded-precision parameter model used in § 4, each independently stored parameter is stored with b bits. For a fixed architecture $\mathcal{A}$, there are constants $K_{\mathcal{A}}, \kappa_{\mathcal{A}}$, depending only on $\mathcal{A}$ and the allowed b-bit parameter range, such that every query–key score of $\mathcal{A}$ on length-i inputs has magnitude at most

$$\Gamma_{\mathcal{A}}(i) := K_{\mathcal{A}}(i+1)^{\kappa_{\mathcal{A}}}. \tag{52}$$

We use the fixed penalty size $M_{\mathcal{A}}(i) := 2\Gamma_{\mathcal{A}}(i) + 1$ in Lemma 14. This penalty is large enough to make every disallowed position lose, but it leaves the scores of allowed positions unchanged.

Allowed positions. The lifted head restricts attention to the positions that would be present in the corresponding imagined input. For the owner lane,

$$J_i^O := \{j \le i :\ x_j \in \Sigma_{\mathrm{ord}},\ \omega_j = \omega_i\}. \tag{53}$$

For the pending lane, the active frame is a concatenation of the host frame and the unfinished suffix. These two pieces use

$$\begin{aligned} J_i^{\mathrm{host}} &:= \{j < \beta_i :\ x_j \in \Sigma_{\mathrm{ord}},\ \omega_j = \mathrm{host}_i\}, \\ J_i^{\mathrm{blk}} &:= \{j :\ \beta_i \le j \le i,\ \beta_j = \beta_i\}. \end{aligned} \tag{54}$$

The set J_i^O is nonempty whenever the owner lane is active, and J_i^{blk} is nonempty whenever $i \in \mathcal{P}$.

Concrete form of one lifted head. Fix layer ℓ and original head a of $\mathcal{A}$. We now spell out how this one original head is called on the two lanes. Its learned block consists of

$$W_Q^{(\ell,a)}, \quad W_K^{(\ell,a)}, \quad W_V^{(\ell,a)}, \quad W_O^{(\ell,a)}, \tag{55}$$

together with the corresponding biases. At layer $\ell - 1$, the lifted architecture maintains two lanes,

$$H^{O,(\ell-1)} \qquad \text{and} \qquad H^{P,(\ell-1)}. \tag{56}$$

The owner application of head (ℓ, a) uses

$$q_i^O := W_Q^{(\ell,a)} H_i^{O,(\ell-1)} + b_Q^{(\ell,a)}, \tag{57}$$

$$k_j^O := W_K^{(\ell,a)} H_j^{O,(\ell-1)} + b_K^{(\ell,a)}, \tag{58}$$

$$v_j^O := W_V^{(\ell,a)} H_j^{O,(\ell-1)} + b_V^{(\ell,a)}. \tag{59}$$

The host part of the pending application uses

$$q_i^P := W_Q^{(\ell,a)} H_i^{P,(\ell-1)} + b_Q^{(\ell,a)}, \tag{60}$$

$$k_j^H := W_K^{(\ell,a)} H_j^{O,(\ell-1)} + b_K^{(\ell,a)}, \tag{61}$$

$$v_j^H := W_V^{(\ell,a)} H_j^{O,(\ell-1)} + b_V^{(\ell,a)}. \tag{62}$$

The suffix-block part of the pending application uses the same pending query and

$$\begin{aligned} k_j^K &:= W_K^{(\ell,a)} H_j^{P,(\ell-1)} + b_K^{(\ell,a)}, \\ v_j^K &:= W_V^{(\ell,a)} H_j^{P,(\ell-1)} + b_V^{(\ell,a)}. \end{aligned} \tag{63}$$

In the unfolded architecture, the owner, host, and suffix-block applications are independent copied blocks. We write them using the same symbols $W_Q^{(\ell,a)}, W_K^{(\ell,a)}, W_V^{(\ell,a)}, W_O^{(\ell,a)}$ only to keep notation light; in the simulator parameter setting, all these copies are assigned the corresponding original values from $\mathcal{A}$.

Here $\mathrm{RHard}_{j\in J}$ denotes rightmost-hard attention restricted to positions in J. When the owner lane is active, the owner output is

$$\begin{aligned} u_i^{O,a} &:= \mathrm{RHard}_{j\in J_i^O} \langle q_i^O, k_j^O \rangle \, v_j^O, \\ z_i^{O,a} &:= W_O^{(\ell,a)} u_i^{O,a} + b_O^{(\ell,a)}. \end{aligned} \tag{64}$$

For the pending lane, we compute the best block candidate:

$$(u_i^K, k_i^K) := \mathrm{RHard}_{j\in J_i^{\mathrm{blk}}} \langle q_i^P, k_j^K \rangle \, (v_j^K, k_j^K). \tag{65}$$

If J_i^{host} is nonempty, we also compute the best host candidate:

$$(u_i^H, k_i^H) := \mathrm{RHard}_{j\in J_i^{\mathrm{host}}} \langle q_i^P, k_j^H \rangle \, (v_j^H, k_j^H). \tag{66}$$

A fixed pointwise merge gate then computes

$$\begin{aligned} s_i^H &:= \langle q_i^P, k_i^H \rangle, \\ s_i^K &:= \langle q_i^P, k_i^K \rangle, \end{aligned} \tag{67}$$

and sets

$$u_i^P := \begin{cases} u_i^H, & s_i^H > s_i^K, \\ u_i^K, & s_i^H \le s_i^K. \end{cases} \tag{68}$$

If J_i^{host} is empty, the fixed gate skips the host candidate and sets $u_i^P := u_i^K$. The pending output of head (ℓ, a) is then

$$z_i^{P,a} := W_O^{(\ell,a)} u_i^P + b_O^{(\ell,a)}. \tag{69}$$

Summing over all heads and applying the copied layer-ℓ FFN blocks to the two lanes gives

$$\begin{aligned} \widetilde{H}_i^O &:= H_i^{O,(\ell-1)} + \sum_a z_i^{O,a}, \\ H_i^{O,(\ell)} &:= \mathsf{FFN}^{(\ell)}(\widetilde{H}_i^O), \end{aligned} \tag{70}$$

and

$$\begin{aligned} \widetilde{H}_i^P &:= H_i^{P,(\ell-1)} + \sum_a z_i^{P,a}, \\ H_i^{P,(\ell)} &:= \mathsf{FFN}^{(\ell)}(\widetilde{H}_i^P). \end{aligned} \tag{71}$$

The owner and pending FFN applications are independent copied blocks in $\mathcal{A}'$, again written as $\mathsf{FFN}^{(\ell)}$ for readability and assigned the same value as the original FFN block in the simulator parameter setting. At positions where a lane is inactive, fixed gates keep that lane at zero. At an ordinary token $x_i \in \Sigma_{\mathrm{ord}}$ with $m_i \neq \perp$ (a call- or return-payload), both lanes update.

Lemma 16 (Layer-wise invariant)**.** *The invariant in Equation* (36) *holds at every layer* $\ell \in \{0, \ldots, L\}$.

Proof. By induction on ℓ. Base ($\ell = 0$): from initialization and the positional features at the simulated positions. Step: for $x_j \in \Sigma_{\mathrm{ord}}$, Lemma 14 applied to the owner application matches the original head on O_j at position $|O_j|$. For $j \in \mathcal{P}$: if $\mathsf{HE}_j = 1$, the host half of P_j is empty and the pending computation reduces to the suffix block, so the block candidate is exactly the rightmost-hard winner on P_j; if $\mathsf{HE}_j = 0$, both halves are non-empty and Lemma 15 applies, with the fixed merge gate implementing the rightmost-hard choice across the host and suffix block. Because the copied attention projections, output projections, and FFN blocks are assigned the same numerical values as the corresponding original blocks of $\mathcal{A}$, the layer update is exactly the original layer of $\mathcal{A}$ run on the relevant imagined input. Hence the invariant is preserved. □

D.4 Readout and Resource Bounds

Proof of Theorem 2. Let $\mathcal{A}$ realize the local recursive rule g with depth L. Construct $\mathcal{A}'$ in three stages: (1) prepend the constant-depth bookkeeping block of Lemma 13; (2) replace each of the L layers of $\mathcal{A}$ by its lifted version (§ D.3), so that Lemma 16 gives the invariant in Equation (36) at $\ell = L$; (3) apply the readout below.

Readout. At the final position n, select

$$z_n := \begin{cases} H^{O,(L)}_{j^*_n}, & m_n = \bot,\ C_n \neq \varnothing,\ j^*_n := \max C_n, \\ H^{P,(L)}_n, & m_n \neq \bot, \\ H^{O,(L)}_1, & m_n = \bot,\ C_n = \varnothing \text{ (sentinel branch).} \end{cases} \tag{72}$$

By Lemma 16 and Lemma 12, $z_n = H^{\mathcal{A},L}_{|\Pi_{\mathrm{rec}}(\mathbf{x})|}(\Pi_{\mathrm{rec}}(\mathbf{x}))$ in the first two cases, and $z_n = H^{\mathcal{A},L}_1(\langle \texttt{bos} \rangle)$ in the third, realizing $g(\epsilon) := g(\langle \texttt{bos} \rangle)$. A rightmost exact-match head copies z_n to position n, and a copied decoder block, assigned the same numerical value as the decoder of $\mathcal{A}$, yields $g(\Pi_{\mathrm{rec}}(\mathbf{x}))$.

Learned-parameter accounting. The lifted architecture does more computation than $\mathcal{A}$, because each layer keeps owner and pending states and applies copied versions of the original heads and FFN blocks to the relevant lanes and subroutines. Under the unfolded convention of § C.1, these copies are counted as independent learned parameters. For the simulator witnessing Theorem 2, the copied parameters are assigned the same numerical values as the corresponding parameters of $\mathcal{A}$.

Each original learned block is copied only a constant number of times by the fixed lifting construction: the owner, host, and suffix-block applications for attention heads, the owner and pending FFN applications, and the final decoder copy. Therefore

$$W(\mathcal{A}') = O(W(\mathcal{A})). \tag{73}$$

The hidden constant depends only on the fixed lifting construction, not on the recursive task, the training sample, or the parameter setting of $\mathcal{A}$. The bookkeeping block, allowed-position sets, exact-match selectors, copy operations, and merge gates are fixed non-learned parts of the architecture.

The depth overhead remains constant:

$$L(\mathcal{A}') = L(\mathcal{A}) + O(1). \tag{74}$$

□

E Proofs for § 4 (IID Generalization)

Throughout this appendix, $\mathcal{D}$ gives probability one to input sequences that can arise during recursive generation, as formalized by Definition 7.

Architecture coding convention. The Transformer appendix specifies architectures by finite data: depth, layer dimensions, numbers of heads, feed-forward dimensions, fixed positional features, and the fixed non-learned operations used by the constructions. Fix any prefix universal decoder U for such finite architecture specifications, for instance a prefix universal Turing machine whose outputs are parsed as such specifications. For each architecture $\mathcal{A}$, let $\mathrm{code}(\mathcal{A})$ be a shortest U-program that outputs $\mathcal{A}$, with ties broken deterministically. A full finite-precision Transformer description consists of $\mathrm{code}(\mathcal{A})$ followed by the b-bit encodings of the $W_{\mathcal{A}}$ independently stored learned parameters.

For a next-token rule f, minimizing over its Transformer realizations gives

$$|f|_\ell := \min_{\mathcal{A},\theta:\, f_{\mathcal{A},\theta}=f} \left(|\,\mathrm{code}(\mathcal{A})| + bW_{\mathcal{A}}\right), \qquad \min\emptyset := +\infty. \tag{75}$$

Here the architecture uses a prefix universal code and each learned scalar is stored in b bits; the simulation computes intermediate activations with exact arithmetic.

This is a Kraft-valid complexity measure: the chosen architecture descriptions form a prefix-free set because they are selected from the prefix domain of U. For each fixed $\mathcal{A}$, there are at most $2^{bW_{\mathcal{A}}}$ parameter strings of length $bW_{\mathcal{A}}$, so the total Kraft mass contributed by this architecture is at most $2^{-|\,\mathrm{code}(\mathcal{A})|}$. Summing over architectures is bounded by one.

Lemma 17 (Simulation Coding Overhead)**.** *Let $\mathcal{A}'$ be the simulator architecture produced from $\mathcal{A}$ in Theorem 2. Then*

$$L_{\mathcal{A}'} = L_{\mathcal{A}} + O(1), \qquad W_{\mathcal{A}'} \le C_W W_{\mathcal{A}}, \tag{76}$$

and

$$|\,\mathrm{code}(\mathcal{A}')| \le |\,\mathrm{code}(\mathcal{A})| + C_{\mathrm{arch}} \tag{77}$$

for constants independent of the task and the training sample. Consequently, if a local rule g is realized by $(\mathcal{A},\theta)$, the simulator gives $(\mathcal{A}',\theta')$ realizing $g\circ\Pi_{\mathrm{rec}}$ on legal recursive prefixes with

$$|\,\mathrm{code}(\mathcal{A}')| + bW_{\mathcal{A}'} \le C\big(|\,\mathrm{code}(\mathcal{A})| + bW_{\mathcal{A}}\big) + C_0. \tag{78}$$

Proof. The agreement on legal recursive prefixes and the depth statement are Theorem 2. For the parameter count, the construction in § D uses only a constant number of copied applications of each original learned block under the unfolded convention; bookkeeping, copy, mask, and merge operations are fixed and non-learned. Hence $W_{\mathcal{A}'} \le C_W W_{\mathcal{A}}$.

For the architecture code, the simulator architecture is obtained from $\mathcal{A}$ by a fixed computable transformation S, so $\mathcal{A}' = S(\mathcal{A})$. Given any U-program for $\mathcal{A}$, a constant-size wrapper can decode $\mathcal{A}$ and output $S(\mathcal{A})$. By universality of U, this gives

$$|\,\mathrm{code}(\mathcal{A}')| \le |\,\mathrm{code}(\mathcal{A})| + C_{\mathrm{arch}}. \tag{79}$$

Combining this with the parameter-count bound gives the final inequality. □

Proof of Corollary 3. Fix a complete-sequence predictor f with finite $|f|_{\ell^{\Pi_{\mathrm{rec}}}}$. By definition of the lifted length, choose a shortest local representative g such that $f = g\circ\Pi_{\mathrm{rec}}$ and $|g|_\ell = |f|_{\ell^{\Pi_{\mathrm{rec}}}}$. Choose a shortest finite-precision Transformer realization $(\mathcal{A},\theta)$ of g. By Lemma 17, the simulator produces a CoT predictor $\tilde{f}$ that agrees with $g\circ\Pi_{\mathrm{rec}}$, and hence with f, on legal recursive prefixes. Moreover,

$$\begin{aligned} |\tilde{f}|_\ell &\le |\,\mathrm{code}(\mathcal{A}')| + bW_{\mathcal{A}'} && (80)\\ &\le C\big(|\,\mathrm{code}(\mathcal{A})| + bW_{\mathcal{A}}\big) + C_0 && (81)\\ &\le C|f|_{\ell^{\Pi_{\mathrm{rec}}}} + C_0. && (82) \end{aligned}$$

□

Proof of Theorem 4. Let $S\subseteq\Omega$ be the set of input sequences that can arise during recursive generation. By assumption, a draw $\mathbf{x}\sim\mathcal{D}$ lies in S with probability one. The complete-sequence target is

$$f(\mathbf{x}) := g(\Pi_{\mathrm{rec}}(\mathbf{x})), \tag{83}$$

and

$$\mathcal{L}_{\mathcal{D}}(h) = \Pr_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f(\mathbf{x})]. \tag{84}$$

Lifted recursive description. Apply Theorem 9 under the lifted description language $\ell^{\Pi_{\mathrm{rec}}}$ with target f and confidence $\delta/2$. Since f itself has lifted length $|f|_{\ell^{\Pi_{\mathrm{rec}}}}$, the MDL output

$$\hat{f}_{\Pi_{\mathrm{rec}}} := \mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f) \tag{85}$$

satisfies, with probability at least $1 - \delta/2$,

$$\mathcal{L}_{\mathcal{D}}(\hat{f}_{\Pi_{\mathrm{rec}}}) \leq \frac{|f|_{\ell^{\Pi_{\mathrm{rec}}}} \ln 2 + \ln(2/\delta)}{m}. \tag{86}$$

CoT description. By Corollary 3, there is a CoT predictor $\tilde{f}$ agreeing with f on legal recursive prefixes and satisfying

$$|\tilde{f}|_{\ell} \leq C\,|f|_{\ell^{\Pi_{\mathrm{rec}}}} + C_0. \tag{87}$$

Since $\mathcal{D}$ is supported on legal recursive prefixes, $\tilde{f}$ and f agree almost surely. Apply Theorem 9 under ℓ with target $\tilde{f}$ and confidence $\delta/2$. On the probability-one event that all training samples are legal recursive prefixes, the learner below has the same feasible set, and hence the same output under the fixed tie-breaking rule, as $\mathsf{MDL}_{\ell}(\mathcal{X}, \tilde{f})$:

$$\hat{f} := \mathsf{MDL}_{\ell}(\mathcal{X}, f), \tag{88}$$

and the loss against $\tilde{f}$ equals the loss against f. Therefore, with probability at least $1 - \delta/2$,

$$\mathcal{L}_{\mathcal{D}}(\hat{f}) \leq \frac{\big(C\,|f|_{\ell^{\Pi_{\mathrm{rec}}}} + C_0\big) \ln 2 + \ln(2/\delta)}{m}. \tag{89}$$

A union bound gives both inequalities simultaneously with probability at least $1 - \delta$. □

F Proofs for Section 5

This appendix contains the supporting results used in § 5. For the general invariance statements, $\Pi : \Omega \to \Omega$ denotes an arbitrary idempotent observation function, $g : \Omega \to \Sigma$ is the local target rule, and the induced target on complete sequences is

$$f(\mathbf{x}) := g(\Pi(\mathbf{x})). \tag{90}$$

Specializing to the recursive model means setting $\Pi = \Pi_{\mathrm{rec}}$, the active-frame observation function from § 3. The set $\mathcal{X} \subseteq \Omega$ always denotes the training set of complete sequences.

F.1 The Witness and the Strict-Gap Condition

Thus a strict shortcut gap certifies non-invariance: the CoT predictor cannot be determined by the visible context alone. The failure is not on an entirely unfamiliar input. For some training sequence $\mathbf{x}$, the CoT predictor is correct on $\mathbf{x}$ but changes its answer on the test input $\mathbf{x}_{\mathrm{te}} := \Pi(\mathbf{x})$. Thus the failure lies in $\Pi(\mathcal{X}) \subseteq \mathrm{Eq}_{\Pi}(\mathcal{X})$: the visible context is familiar, but the surrounding complete sequence has changed.[2]

[2]The strict gap is sufficient, not necessary. If $\Pi(\mathcal{X}) \subseteq \mathcal{X}$, then by idempotence any CoT predictor consistent on $\mathcal{X}$ also makes $\hat{f} \circ \Pi$ feasible for the lifted MDL problem with $|\hat{f} \circ \Pi|_{\ell^{\Pi}} \leq |\hat{f}|_{\ell}$, so a strict shortcut gap cannot occur. Even without such a gap, CoT can still choose a non-invariant rule, for instance through a tie or another consistent choice (Proposition 18).

F.2 Strict Gap Is Not Necessary

Theorem 6 gives a sufficient condition for non-invariance: the CoT predictor is strictly shorter under ℓ than the lifted MDL output under ℓ^{Π}. The condition is not necessary. A learner that reads the complete sequence can choose a non-invariant predictor even when the lifted MDL problem has an equally short or shorter solution.

Proposition 18 (Non-Invariance Without a Strict Shortcut Gap)**.** *There exist a local recursive target g and its induced full-sequence target $f = g \circ \Pi_{\mathrm{rec}}$, a complexity measure ℓ, and a training set of complete sequences $\mathcal{X} \subseteq \Omega$ such that the CoT MDL output*

$$\hat{f} := \mathsf{MDL}_{\ell}(\mathcal{X}, f) \tag{91}$$

is not Π_{rec}-invariant on $\mathcal{X}$, but the strict shortcut gap fails:

$$|\hat{f}|_{\ell} \geq |\mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f)|_{\ell^{\Pi_{\mathrm{rec}}}} . \tag{92}$$

Proof. Choose a control-free string $\mathbf{v} \in \Sigma^*_{\mathrm{ord}}$ and let

$$\mathbf{x}_0 := \langle \texttt{call} \rangle \, \mathbf{v} \, \langle / \texttt{call} \rangle. \tag{93}$$

Under the stack procedure, $\mathbf{x}_0$ opens a child frame initialized with $\mathbf{v}$, so

$$\Pi_{\mathrm{rec}}(\mathbf{x}_0) = \mathbf{v}, \qquad \Pi_{\mathrm{rec}}(\mathbf{v}) = \mathbf{v}. \tag{94}$$

Let $\mathcal{X} := \{\mathbf{x}_0\}$. Pick two distinct output tokens $a_0, a_1 \in \Sigma$, and define the local target g by $g(\mathbf{v}) = a_0$; its values elsewhere are arbitrary. Then $f(\mathbf{x}_0) = a_0$ and $f(\mathbf{v}) = a_0$.

Now define three hypotheses:

$$h_{\mathrm{non}}(\mathbf{u}) := \begin{cases} a_0, & \mathbf{u} = \mathbf{x}_0, \\ a_1, & \text{otherwise}, \end{cases} \tag{95}$$

$$h_{\mathrm{loc}}(\mathbf{u}) := \begin{cases} a_0, & \mathbf{u} = \mathbf{v}, \\ a_1, & \text{otherwise}, \end{cases} \tag{96}$$

$$h_{\mathrm{inv}}(\mathbf{u}) := \begin{cases} a_0, & \Pi_{\mathrm{rec}}(\mathbf{u}) = \mathbf{v}, \\ a_1, & \text{otherwise}. \end{cases} \tag{97}$$

Assign finite description lengths

$$|h_{\mathrm{loc}}|_{\ell} = 1, \qquad |h_{\mathrm{non}}|_{\ell} = 2, \qquad |h_{\mathrm{inv}}|_{\ell} = 3. \tag{98}$$

All other predictors have length $+\infty$. These lengths satisfy the Kraft inequality, since $2^{-1} + 2^{-2} + 2^{-3} < 1$.

For the CoT learner, feasibility means agreeing with f directly on $\mathcal{X}$. Both h_{non} and h_{inv} are feasible, but h_{loc} is not because $h_{\mathrm{loc}}(\mathbf{x}_0) = a_1$. Hence the CoT MDL output under the original description language is

$$\hat{f} = h_{\mathrm{non}}, \qquad |\hat{f}|_{\ell} = 2. \tag{99}$$

This predictor is not Π_{rec}-invariant on $\mathcal{X}$ because

$$\Pi_{\mathrm{rec}}(\mathbf{x}_0) = \Pi_{\mathrm{rec}}(\mathbf{v}) = \mathbf{v}, \qquad h_{\mathrm{non}}(\mathbf{x}_0) = a_0 \neq a_1 = h_{\mathrm{non}}(\mathbf{v}). \tag{100}$$

It also fails on the OOD sequence $\mathbf{v}$, since $f(\mathbf{v}) = a_0$ while $h_{\mathrm{non}}(\mathbf{v}) = a_1$.

For the lifted learner, we use $\ell^{\Pi_{\mathrm{rec}}}$ on complete-sequence predictors. The invariant predictor satisfies

$$h_{\mathrm{inv}} = h_{\mathrm{loc}} \circ \Pi_{\mathrm{rec}}, \qquad h_{\mathrm{inv}}(\mathbf{x}_0) = a_0 = f(\mathbf{x}_0), \tag{101}$$

so h_{inv} is feasible for $\mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f)$. Moreover

$$|h_{\mathrm{inv}}|_{\ell^{\Pi_{\mathrm{rec}}}} \leq |h_{\mathrm{loc}}|_{\ell} = 1. \tag{102}$$

No feasible complete-sequence predictor can have lifted length below 1, because the only finite-length local representative with length 1 is h_{loc}, and it induces h_{inv}. Thus

$$\mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f) = h_{\mathrm{inv}}, \qquad |\mathsf{MDL}_{\ell^{\Pi_{\mathrm{rec}}}}(\mathcal{X}, f)|_{\ell^{\Pi_{\mathrm{rec}}}} = 1. \tag{103}$$

Thus the strict shortcut gap does not hold: $2 < 1$ is false. Nevertheless, the CoT MDL output is non-invariant and makes an OOD error. This proves that the strict gap is sufficient, but not necessary, for the failure mode described in § 5. □

G Experimental Details

G.1 Function Library

The expression generator draws from the typed function library in Table 3. Primitive functions are evaluated directly. Composite functions are named bodies that expand into calls to other library functions. Tail-recursive functions are also named bodies, but their bodies may call the same function again with updated arguments. This last group is what lets a shallow written expression produce a much longer call/return trace at execution time. Integer-valued outputs are reduced modulo 10; Boolean-valued outputs are emitted as Boolean tokens.

Table 3: Function library used by the expression generator.

Function	Type	Body / definition
Primitive functions		
`add`	int, int → int	$(x + y) \bmod 10$
`sub`	int, int → int	$(x - y) \bmod 10$
`multiply`	int, int → int	$(x \cdot y) \bmod 10$
`diff`	int, int → int	$\lvert x - y \rvert \bmod 10$
`square`	int → int	$x^2 \bmod 10$
`double`	int → int	$2x \bmod 10$
`min`	int, int → int	$\min(x, y)$
`max`	int, int → int	$\max(x, y)$
`less`	int, int → bool	$\mathbf{1}\{x < y\}$
`is_even`	int → bool	$\mathbf{1}\{x \bmod 2 = 0\}$
`if_then_else`	bool, τ, $\tau \to \tau$	returns the second argument if the condition is true, otherwise the third
Composite functions		
`triple_add`(a, b, c)	int, int, int → int	`add`(`add`(a, b), c)
`sum_of_squares` (a, b)	int, int → int	`add`(`square`(a), `square`(b))
`diff_of_squares` (a, b)	int, int → int	`sub`(`square`(a), `square`(b))
`clamp`(v, l, u)	int, int, int → int	`min`(`max`(v, l), u)
`manhattan` (x_1, y_1, x_2, y_2)	int, int, int, int → int	`add`(`diff`(x_1, x_2), `diff`(y_1, y_2))
`is_in_range`(v, l, u)	int, int, int → bool	`less`(`sub`(v, l), `sub`(u, l))
`point_in_rect` $(x, y, x_1, y_1, x_2, y_2)$	six int → bool	`if_then_else`(`is_in_range`(x, x_1, x_2), `is_in_range`(y, y_1, y_2), `False`)
Tail-recursive functions		
`accum_sum`(n, acc)	int, int → int	`if_then_else`(`less`$(n, 1)$, acc, `accum_sum`(`sub`$(n, 1)$, `add`(acc, n)))
`factorial`(n, acc)	int, int → int	`if_then_else`(`less`$(n, 2)$, acc, `factorial`(`sub`$(n, 1)$, `multiply`(acc, n)))
`fibonacci`(n, a, b)	int, int, int → int	`if_then_else`(`less`$(n, 1)$, a, `fibonacci`(`sub`$(n, 1)$, b, `add`(a, b)))

Example expansions. For example,

$$\begin{aligned} \texttt{sum_of_squares}(a, b) &\Rightarrow \texttt{add}(\texttt{square}(a), \texttt{square}(b)), \\ \texttt{accum_sum}(n, \text{acc}) &\Rightarrow \texttt{if_then_else}\big(\texttt{less}(n, 1),\ \text{acc}, \\ &\qquad \texttt{accum_sum}(\texttt{sub}(n, 1),\ \texttt{add}(\text{acc}, n))\big). \end{aligned} \tag{104}$$

Worked trace. We illustrate how a written expression turns into the two training views using the example `add`(`sum_of_squares`$(2, 3)$, `double`(4)), whose ground-truth value is 1. Table 4 shows

the complete flattened trace on the left and the active-frame decomposition on the right. This execution has 100 flattened tokens under the tokenizer from § G.2, 11 active-frame examples, and maximum stack depth 4. The example is small enough to print, but it already shows the main mechanism: the composite call `sum_of_squares`$(2, 3)$ expands into `add`$($`square`$(2), $`square`$(3))$, and those nested calls create separate frames in the execution trace. The two views encode the same execution but expose different information at training time:

- The CoT learner trains on the entire flattened trace as one long sequence, so at every prediction step the model sees *all* earlier tokens of the trace.
- The RM learner trains on the eleven active-frame examples as separate examples. Each example contains the visible context (the parent tokens before the boundary m) followed by what is generated inside that frame. Loss is applied only to the latter. The RM learner thus never sees, at the same training step, tokens belonging to a sibling frame or a popped child frame.

In the right-hand column, d is stack depth and m is the visible-context boundary for that frame.

Table 4: Worked trace: flattened CoT view and active-frame RM view.

(A) Flattened trace (CoT view)	**(B) Active-frame decomposition (RM view)**
add (sum_of_squares (2 , 3) , double (4)) x := <call> sum_of_squares (2 , 3) </call> a := 2 b := 3 return <return> <call> add (square (2) , square (3)) </call> x := <call> square (2) </call> x := 2 return <return> 4 </return> y := <call> square (3) </call> x := 3 return <return> 9 </return> return <return> 3 </return> </return> y := <call> double (4) </call> x := 4 return <return> 8 </return> return <return> 1 </return>	**F1** (d=1, m=8): add (sum_of_squares (2 , 3) | , double (4)) x := <call> sum_of_squares (2 , 3) </call> **F2** (d=2, m=6): sum_of_squares (2 , 3) | a := 2 b := 3 return <return> <call> add (square (2) , square (3)) </call> **F3** (d=3, m=12): add (square (2) , square (3)) | x := <call> square (2) </call> **F4** (d=4, m=4): square (2) | x := 2 return <return> 4 </return> **F5** (d=3, m=15): add (square (2) , square (3)) x := 4 | y := <call> square (3) </call> **F6** (d=4, m=4): square (3) | x := 3 return <return> 9 </return> **F7** (d=3, m=18): add (square (2) , square (3)) x := 4 y := 9 | return <return> 3 </return> **F8** (d=2, m=15): sum_of_squares (2 , 3) a := 2 b := 3 return <return> 3 | </return> **F9** (d=1, m=17): add (sum_of_squares (2 , 3) , double (4)) x := 3 | y := <call> double (4) </call> **F10** (d=2, m=4): double (4) | x := 4 return <return> 8 </return> **F11** (d=1, m=20): add (sum_of_squares (2 , 3) , double (4)) x := 3 y := 8 | return <return> 1 </return>

The vertical bar | in the right-hand column marks the mask-index boundary m: the visible context (in blue) is shown to the recursive learner but receives no loss signal; the generated tokens to its right are the loss-bearing targets for that frame.

G.2 Experimental Setup

This section records the setup shared by all experiments. The three following appendix subsections describe the parts that differ across the IID, length/depth, and shortcut studies.

Task and generator. The task is symbolic expression evaluation. Expressions are built from a typed function library, listed in § G.1. Integer-valued operations are evaluated modulo 10, while Boolean-valued operations return Boolean tokens. The library contains one-step primitives, shallow composite functions, and tail-recursive functions that can repeatedly call themselves during execution.

To sample an expression, we choose the root uniformly from the available function pool. Each argument is then filled independently: with probability $p_{\text{lit}} = 0.4$ we draw a digit literal uniformly

from $\{0, \ldots, 9\}$, and otherwise we recursively draw a type-compatible sub-expression. We enforce type consistency throughout, so Boolean arguments are filled only by Boolean-returning expressions, and we stop the static recursion at nesting depth 3. This bound applies to the written expression, not to the executed trace: tail-recursive functions can unroll at run time and produce much longer traces and deeper stacks. The experiment-specific sections below describe how these generated expressions are split into training, IID validation, and OOD test sets.

Tokenization and traces. We use a shared whitespace tokenizer over function names, digit literals, syntax tokens, Boolean literals, end-of-sequence markers, and the four trace control tokens $\{\langle\texttt{call}\rangle, \langle/\texttt{call}\rangle, \langle\texttt{return}\rangle, \langle/\texttt{return}\rangle\}$. Executing an expression produces a canonical call/return trace. A $\langle\texttt{call}\rangle$ block opens a subproblem; an $\langle\texttt{return}\rangle$ block returns its value to the parent frame. We measure trace length as the number of generated tokens in this canonical trace, including syntax, literal, and control tokens. The maximum recursion depth is the largest stack depth reached during execution, with the root frame counted as depth 1.

Training formats. CoT and RM train on the same underlying expressions but receive different views of the corresponding trace. CoT trains on the complete flattened trace with standard left-to-right next-token prediction. RM trains on active-frame examples: at each prediction step, the visible context is the current frame, and the loss is applied only to tokens generated inside that frame. Thus RM uses the same target computation as CoT, but hides tokens belonging to other active or completed frames. One expression yields one CoT trace example but many RM active-frame examples. The experiment-specific sections state whether the comparison is matched by underlying expressions, by training range, or by another criterion.

Models and optimization. Unless otherwise stated, both learners use the same decoder-only Transformer block: 6 layers, 6 attention heads, embedding dimension 384, MLP expansion factor 4, rotary positional embeddings (RoPE), GELU activations, and pre-LayerNorm. Both models have about 10.65M parameters. In all comparisons, the two learners are trained on the same expression pools; the matching criterion is specified in the relevant experiment subsection.

We train both learners with AdamW. For the length/depth and shortcut experiments, we train until held-out in-distribution accuracy has plateaued for both learners; the IID sample-complexity experiment instead varies the number of training samples and uses the best checkpoint found for each sample size.

Evaluation. At test time, each model generates a trace in its native format. CoT generates a flat trace autoregressively. RM uses a stack driver: when it emits a call, a new active frame is pushed; when it emits an answer, the returned value is written back to the parent frame. We extract the final returned value and score it against the ground-truth expression value: a digit for integer-valued tasks and a Boolean token for Boolean-valued tasks. RM and CoT are evaluated on the same test expressions within each split, and confidence intervals, when reported, use the Wilson score method. The IID sample-complexity experiment also reports a teacher-forced per-token test accuracy diagnostic, described separately in § G.3.

G.3 Detailed Setup for IID Compositional Generalization

Split. § 6.1 uses an IID random split of the expression generator from § G.2. From a seed-42 expression pool, we reserve 5,000 held-out expressions for testing and 1,000 for validation; the remaining expressions form the training pool. Train and test expressions are sampled from the same generator, so this experiment tests generalization to new compositions rather than extrapolation in trace length or recursion depth.

Sample-size sweep. We vary the number of training expressions

$$N \in \{100, 200, 300, 500, 1000, 2000, 3000, 5000, \\ 10000, 20000, 30000, 50000, 100000, 200000, 300000, 450000\}.$$

The subsets are nested: the training set for a smaller N is contained in the training set for a larger N. For each N, RM and CoT train on the same underlying expressions in their native formats. RM uses

active-frame subsequence examples; CoT uses the flattened execution trace. Both use the common 6-layer Transformer backbone.

Training and evaluation. For each (N, format), training is allowed to make multiple passes over the fixed N examples. Training is timeout-based: at regular intervals we run greedy validation generation on IID held-out expressions and save the checkpoint with the best validation final-answer accuracy. The selected checkpoint is then evaluated on the held-out test split. We use three seeds for the smaller sample sizes and one seed for the largest sample sizes.

The left panel of Figure 3 reports final-answer test accuracy from greedy autoregressive generation. The end-to-end sweep has test results through $N = 300{,}000$; the per-token sweep also includes $N = 450{,}000$. The right panel reports teacher-forced per-token test accuracy: at each eligible target position, the model sees the ground-truth prefix and we compare the argmax next-token prediction with the true next token. For CoT, eligible positions are the non-padded target positions of the flattened trace after the prompt. For RM, eligible positions are the loss-bearing active-frame positions at or after the frame's mask boundary. The per-token confidence bands use a sample-level bootstrap over test expressions rather than treating all tokens as independent. The validation split is used only for checkpoint selection; both reported metrics are computed post hoc on the test split.

G.4 Detailed Setup for Length and Depth Generalization

Threshold splits. § 6.2 uses the same generator as the IID experiment, but splits examples by realized execution properties. For a length threshold L, training and validation expressions have generated trace length $\ell \leq L$, while OOD test expressions have $\ell > L$. Here ℓ is executed trace length: the number of tokens in the executed call/return trace, not the prompt length. For a depth threshold K, the same construction is applied to maximum recursion depth $d_{\max}$. Both ℓ and $d_{\max}$ are measured after executing the expression, not from the static written expression.

We draw the sweep data from a fixed 500k-expression pool generated with seed 42. We sweep eight length thresholds $L \in \{33, 96, 197, 359, 595, 1060, 1345, 1839\}$ and five depth thresholds $K \in \{3, 5, 7, 9, 10\}$. The main-text table uses the representative splits $L = 595$ and $K = 10$, and held-out examples from the training side of each threshold provide the IID check.

For Table 1, we further group evaluation expressions by their multiplier relative to the split threshold: ℓ/L for the length split and $d_{\max}/K$ for the depth split. The reported bins are fixed multiplier intervals, not quantiles. The representative length and depth tables each use $3{,}100$ evaluation expressions shared by RM and CoT; the smallest reported bins contain 72 and 159 examples, respectively.

Training and evaluation. For each split, RM and CoT use the common architecture and training protocol from § G.2. We train both learners until held-out below-threshold validation accuracy plateaus at at least 99%, then evaluate final-answer accuracy on OOD expressions from the corresponding test side. All accuracy numbers in Table 1 use the same expression samples for RM and CoT within each bin.

Frame-width diagnostic. The length split has one additional diagnostic. For an expression, let $w_{\max}$ be the largest token length of any active frame that RM must process during execution. Let

$$T_{\max} = \max_{x \in \mathcal{X}_{\text{train}}} w_{\max}(x) \tag{105}$$

be the largest active-frame width observed on the training side of the split. Filtering OOD examples by $w_{\max} \leq T_{\max}$ asks whether RM's remaining errors come from genuinely longer traces, or from active frames that are themselves wider than anything seen in training.

For the representative length split, $T_{\max} = 115$. The filter matters only in the farthest length bin: for traces longer than $5L$, RM accuracy rises from 76.4% to 85.0% on the retained subset. For the representative depth split, $T_{\max} = 160$ and the filter retains all samples, so the depth shift is not a single-frame-width artifact. In other words, the final RM drop in the length table partly reflects active frames that are wider than any training frame, whereas the depth table isolates a shift in the number of nested frames. Since the unfiltered numbers are the primary result, we report this diagnostic in prose rather than as an additional table.

G.5 Detailed Setup for Shortcut Behavior

Train/OOD construction. The shortcut experiments use paired train/OOD distributions. Each construction defines a shortcut signal $S(\mathbf{x})$ that is visible somewhere in the complete CoT trace but hidden from the active frame visible to RM. In training, the data are constructed so that $S(\mathbf{x})$ agrees with the answer. In OOD evaluation, this coupling is broken while the intended root computation remains valid. The constructed cases may introduce fixed wrapper or helper templates at the root; the recursive sub-expressions inside those templates are sampled from the library in § G.1. Except where stated otherwise, the OOD split keeps the same distribution over recursive sub-expressions and changes only the argument or filter that made the shortcut agree with the answer. Thus IID accuracy checks whether the distribution is learnable, and OOD accuracy plus shortcut rate checks which rule the model learned.

Let $\widehat{\mathrm{ans}}_{\mathrm{CoT}}(\mathbf{x})$ be CoT's final returned value. On an OOD sample $\mathcal{X}_{\mathrm{OOD}}$, the shortcut rate is

$$\mathrm{follow}_S := \frac{1}{|\mathcal{X}_{\mathrm{OOD}}|} \sum_{\mathbf{x} \in \mathcal{X}_{\mathrm{OOD}}} \mathbf{1}\{\widehat{\mathrm{ans}}_{\mathrm{CoT}}(\mathbf{x}) = S(\mathbf{x})\}. \tag{106}$$

All six constructions in Table 2 are evaluated on $n = 300$ IID and $n = 300$ OOD expressions.

Case 1: copy answer from a deep return. Let Z be a deep recursive sub-expression with return value z, drawn from the tail-recursive part of the library. The training root template is

$$\texttt{add}(\texttt{square}(\texttt{square}(Z)),\, Y_{\mathrm{comp}}), \qquad Y_{\mathrm{comp}} := (z - z^4) \bmod 10. \tag{107}$$

The training answer is therefore z, so $S(\mathbf{x}) = z$. OOD resamples the second argument, so the answer is computed from the root-visible value $z^4 \bmod 10$ and the new argument. CoT can read the deep return token z; RM sees only the transformed root payload. The double-square wrapper is deliberately non-injective modulo 10, so RM cannot recover z from the root-visible payload by a one-to-one lookup.

Case 2: trace-position parity. The root template is $\texttt{is_even}(\texttt{add}(Z, Y))$. The shortcut signal $S(\mathbf{x})$ is the parity of the final answer token's position in the complete trace, encoded as the binary label. Training keeps examples where this parity agrees with the label; OOD keeps examples where the relationship is flipped. The marginal distributions of Z and Y are kept the same across the two splits; only the coupling between label and global answer position changes. CoT can read the global position signal from the flat trace, while RM sees only local frame positions because every active frame starts from its own local offset.

Hidden intermediate literal. This construction hides a runtime literal inside an intermediate helper frame. A helper expression first evaluates a sub-expression W, computes $k = \texttt{eval}(W) \bmod 10$, and tail-calls another subproblem with k as one of its arguments. The root wraps this helper inside a selector that discards the helper's returned value before the payload reaches the root. Thus the literal k appears in the flat trace, but not in the static root expression and not in the root-frame payload visible to RM. Training uses a compensating argument so that the answer equals k; OOD keeps the same hidden computation but resamples the compensating argument, so the correct root answer no longer equals k.

Long-range aggregate. The root template is $\texttt{add}(Z, Y_{\mathrm{comp}})$. Let r_1, r_2, r_3 be the first three integer returns appearing inside Z's trace, and define

$$S(\mathbf{x}) = (r_1 + r_2 + r_3) \bmod 10. \tag{108}$$

If z is the final return value of Z, training chooses $Y_{\mathrm{comp}} = (S(\mathbf{x}) - z) \bmod 10$ so that the root answer equals $S(\mathbf{x})$. OOD resamples the second argument, breaking this equality. CoT can read the three deep return tokens from the flat trace; RM sees only the final return value z of Z at the root. We build the candidate pool for Z by rejection sampling so that the same final value z occurs with different shortcut values S. This prevents RM from recovering the shortcut from the root payload alone.

Near-answer aggregate. This construction is the same as the long-range aggregate, except that S uses the last three internal returns of Z rather than the first three. The shortcut is still hidden from the root frame, but it is more local in the flat trace because the relevant returns occur near the answer-generation region. This tests whether CoT follows a shortcut even when the hidden signal is close to the final answer rather than buried early in the trace.

Position-weighted aggregate. Again using the root template $\texttt{add}(Z, Y_{\text{comp}})$, the shortcut signal is

$$S(\mathbf{x}) = \left(\sum_{t: r_t = 0} t \right) \bmod 10, \tag{109}$$

where r_t is the internal return value at trace position t inside Z. Training chooses the compensating argument so that the answer equals this position-weighted aggregate; OOD resamples the argument and breaks the relationship. This case makes the shortcut depend on both values and global positions in the flat trace.

Validity conditions. The constructions are designed to isolate shortcut visibility rather than artifacts of the generator. Two conditions are important. First, the intended root-frame computation must be identifiable from RM's training view; otherwise RM could fail for reasons unrelated to hidden shortcuts. Second, the root-frame payload must not give a one-to-one code for S; otherwise RM could recover the shortcut without seeing the hidden trace. We also exclude variants where S appears as a literal in the static root expression. The six constructions in Table 2 satisfy these conditions.

Two ablations motivated these checks. In sibling-copy variants, the root frame itself underdetermines which child value should be returned, and RM can fail even though the shortcut is not visible. In one-to-one wrapper variants of Case 1, the root payload contains an invertible code for the hidden value z, so RM can learn the same shortcut from its own view. We exclude both types because they no longer isolate the difference between full-trace visibility and active-frame visibility.

H Additional Shortcut Examples

The two examples below supplement the recursive-trace example in Figure 2. They illustrate how a training domain can make a simpler rule indistinguishable from the intended target.

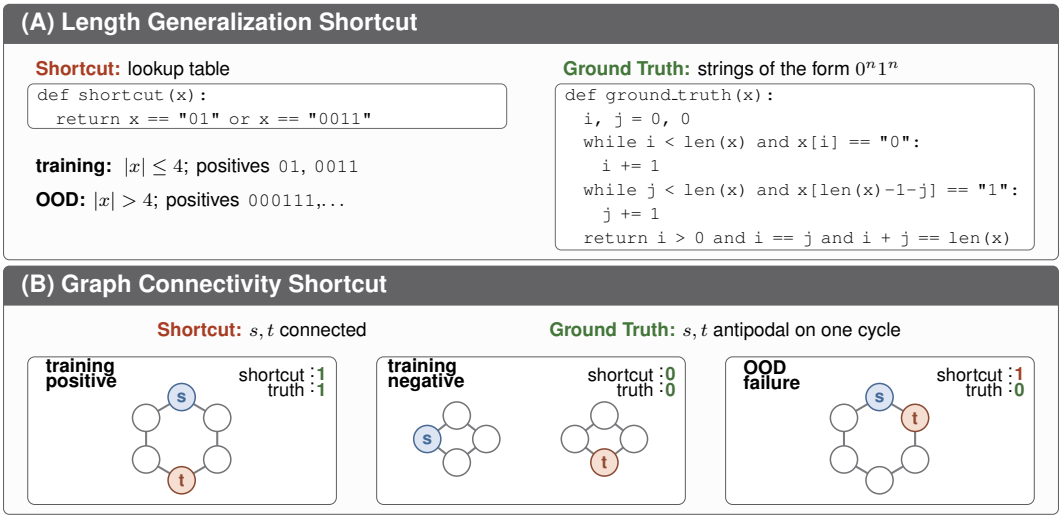

Figure 4: **Shortcut mechanisms.** A simpler rule fits the training domain but fails once the training-domain correlation is broken. The string and graph examples illustrate this possibility outside recursive computation.

In length generalization, let $f^*(x) = 1$ iff $x = 0^n1^n$ for some $n \geq 1$. If the training set is $\mathcal{X}_4 = \{0,1\}^{\leq 4}$, the lookup rule $r_{\text{lookup}}(x) = \mathbb{1}\{x \in \{01, 0011\}\}$ agrees with every training input but rejects every longer balanced string. The intended rule must check the full pattern 0^n1^n: a nonempty block of 0s, followed by a block of 1s of the same length. On this small training set, listing the two positive strings can be shorter than implementing the general rule, so MDL can prefer the lookup even though it fails on the intended test inputs.

For graph classification, suppose positives in the training domain are cycles where marked nodes s, t are antipodal, and negatives are two disjoint cycles with s, t in different components. The shortcut rule "are s, t connected?" agrees with all training inputs. The intended rule is more specific: it must check that s, t lie on the same cycle and that their distance is exactly half the cycle length. Connectivity is a coarser and usually simpler predicate, but it fails on connected graphs outside the training domain where s, t are not antipodal. Again, the shortcut is not nonsense; it is a simpler rule that happens to agree with the training inputs.