A NEURO-SYMBOLIC APPROACH TO LENGTH GENERALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Length generalization (LG), which is the ability of a model trained on short inputs to correctly reason over substantially longer inputs, is a fundamental and challenging problem in learning to reason. Although chain-of-thought (CoT) training can improve LG, the challenge remains. Recent approaches have also introduced sophisticated positional tags to bias the model toward more structured reasoning. While effective to some extent, they still struggle to generalize to substantially longer inputs. In this paper, we propose a novel neuro-symbolic approach to LG for a class of reasoning problems, whose CoT input can use tags to indicate the positions of tokens involved in each reasoning step. The neural training attends only to these positions and their immediate neighbors, together with bounded control information. Instead of directly predicting the output sequence as in existing methods, the proposed neural model predicts only a fixed set of actions that are independent of input length. The actions are executed symbolically outside the neural model to produce the next state. Due to the fixed local contexts and action set, the proposed method is length-invariant and can be applied to inputs of arbitrary length for the class of problems. We evaluate the proposed approach on 10 reasoning tasks, including parity, addition, subtraction, multiplication, and division. Despite training on short sequences, our method consistently generalizes to sequences of very long lengths, achieving perfect LG across the evaluated tasks.

1 INTRODUCTION

Length generalization (LG), namely, the ability of a model trained on short inputs to correctly solve problems with substantially longer inputs, is a fundamental challenge in learning to reason. In practice, neural models, including Transformers (Vaswani et al., 2017), often struggle to extrapolate beyond the training lengths (Zhou et al., 2024b; Press et al., 2022). Although Chain-of-thought (CoT) (Wei et al., 2022) or Scratchpad (Nye et al., 2021), which exposes intermediate reasoning steps during training, can substantially improve LG (Anil et al., 2022; Lee et al., 2024), the challenge remains (Zhou et al., 2024a; Xiao & Liu, 2025; Dziri et al., 2023; Kazemnejad et al., 2023). Recent studies have also investigated how to improve LG by incorporating additional structural information into Transformer reasoning. For example, alternative positional representations have been shown to affect the ability of Transformers to extrapolate to longer sequences (Press et al., 2022; Jelassi et al., 2023; Zhao et al., 2024). More recently, sophisticated positional tags or related mechanisms have been introduced to explicitly bias the model toward more structured reasoning (Cho et al., 2025; 2024; McLeish et al., 2024; Xiao & Liu, 2025). Such approaches have substantially improved LG.

In this paper, we take a different perspective on LG. We argue that a fundamental difficulty of LG is that conventional neural reasoning methods directly predict a sequence of tokens representing the reasoning process. Such a representation makes the learned computation closely coupled to the length and token positions. We ask a more basic question: *What needs to be predicted at each reasoning step for the same reasoning procedure to be applicable to inputs of arbitrary length?*

This means that the learning needs to achieve *length-invariance*. We achieve it through a novel *neuro-symbolic* framework for a class of reasoning problems, for which we can use a set of tags to indicate the positions of tokens involved in each reasoning step. For the neural model, we use the Transformer. During training, the Transformer attends only to these positions and their immediate neighbors, and learns to predict only the local result at the tagged positions and the relative tag-

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.

movement actions required to construct the next state or reasoning step. We put all these as a fixed set of actions and call it the action vector. The actions are performed deterministically by a symbolic component outside of the neural model to produce the next reasoning state or step. For example, to add 345 and 678, the problem is transformed $345_I + 678_J$, where tags I and J indicate the tokens (5 and 8) involved in the next calculation step. The neural model predicts the sum of the two tokens, including the resulting digit 3 and the carry 1, and also that for the next step, I and J both need to be moved left by one position, which is performed by the external symbolic component to give $34_I5 + 67_J8$. For more complex tasks, the neural model can also predict whether a learned child policy should be invoked, while the symbolic component executes the invocation. For example, the Multiplication policy can invoke a learned Addition policy to perform intermediate additions.

Since (1) the Transformer is restricted to attend only to the tagged positions and their immediate left and right neighbors, together with a fixed amount of control information, collectively referred to as the *local context*, and (2) the model predicts only a set of actions that, for the class of problems considered, can be determined solely by this local context, the proposed framework is *length-invariant*. Consequently, the learned model, or policy, can be applied to inputs of arbitrary length within this class of reasoning problems.

We evaluate our approach on 10 reasoning tasks, including parity, addition, subtraction, multiplication, and division. The model is trained on short sequences (1-8) and evaluated on substantially longer sequences. Across all tasks, our approach achieves perfect accuracy on these long sequences, demonstrating that representing reasoning as length-independent local actions, together with symbolic execution of those actions, provides a simple and effective approach to LG.

Our contributions are summarized as follows:

- We propose a neuro-symbolic framework for LG that combines a neural predictor of state-transition actions with a symbolic executor that applies the predicted actions to construct the next state. This approach differs significantly from existing LG methods, which typically train models to directly predict the entire sequence representing the next state. To the best of our knowledge, this is the first neuro-symbolic approach to LG.
- The neural model, implemented as a Transformer, learns only from *local contexts* and predicts only actions that can be determined from these contexts. This defines a class of reasoning problems that satisfy these conditions. For this class, the proposed learning is *length-invariant*, and the learned policy can be applied to inputs of arbitrary length.
- Our system achieves perfect LG on 10 challenging reasoning problems over input lengths far beyond those encountered during training.

2 Related Work

The challenge of length generalization (LG) (Anil et al., 2022; Zhou et al., 2024a) reflects a fundamental challenge of conventional machine learning: under the standard i.i.d. learning paradigm, models are primarily optimized to generalize to data drawn from the same distribution as the training data, rather than to extrapolate systematically beyond the training range. For example, Hu et al. (2024) showed that Transformers may solve reasoning tasks through case-based rather than rule-based reasoning, relying on similarities to examples encountered during training instead of learning and applying the underlying rules. Several theoretical studies have also investigated the learnability of LG on Transformer (Abbe et al., 2024; Awasthi et al., 2025; Huang et al., 2025; Izzo et al., 2026). However, their empirical results are not satisfactory.

In the past few years, many approaches have been proposed to improve LG. Among the most common are methods that modify positional embeddings or apply various calibration techniques (Duan et al., 2023; Kazemnejad et al., 2023; Jelassi et al., 2023; Sun et al., 2023; Press et al., 2022; Ruoss et al., 2023; Han et al., 2024; He et al., 2024; Li et al., 2024; Sabbaghi et al., 2024). Jelassi et al. (2023) further showed that incorporating a small number of longer examples during training can improve LG. Chi et al. (2023) introduced a modified Transformer architecture that incorporates weight sharing, working memory, and other architectural mechanisms to improve LG on regular languages. Other studies have investigated the recognition of formal languages (Deletang et al., 2023; Butoi et al., 2025), while Nguyen et al. (2026) proposed combining DeltaNet with a stack to address LG.

While these prior works have demonstrated improved performance in their respective settings and reasoning tasks, they remain far from achieving robust LG. In particular, they continue to face substantial difficulties with more challenging arithmetic tasks, such as long addition and even relatively short multiplication. Recently, the LG system TAIL (Hua et al., 2026) was also evaluated on the Division task, but its accuracy decreased substantially at longer lengths. To our knowledge, no prior work has demonstrated perfect LG for large-number multiplication or Division.

Many other methods train models to map inputs directly to final outputs without explicitly modeling intermediate reasoning steps or CoT. They are mainly based on alternative attention mechanisms and architectural designs (Tay et al., 2021; Chowdhury & Caragea, 2023a;b). As mentioned above, theoretical studies have established strong limitations on learning such reasoning tasks without intermediate computational steps (Wies et al., 2023; Feng et al., 2023).

Our work is closely related to (n, r)-*consistency* (Xiao & Liu, 2025) and *positional coupling* (Cho et al., 2025; 2024; McLeish et al., 2024). They both incorporated additional structural information into positional representations. Specifically, they introduce positional tags to provide additional inductive bias, thereby improving their ability to achieve length generalization (LG). However, these methods are complex and require the design of numerous task-specific tags. More importantly, neither provides a systematic procedure for choosing such tags and where to place them. For example the method in (Cho et al., 2025) needs to provide one tag for each token position and its addition and multiplication still give weak LG results. The method in (Xiao & Liu, 2025) needs 3 tags in each step of addition, 9 tags for each step of multiplication or division. Our approach also uses positional tags, but in a substantially simpler and more systematic manner. Specifically, our tags merely mark the token positions involved in each calculation step (needing only 3 tags for addition, multiplication or division), without requiring any other additional, complex task-specific tags introduced by these prior approaches. The method in (Xiao & Liu, 2025) still struggles with substantially long inputs.

3 METHOD

Overview: The proposed method is based on a neuro-symbolic framework. The *neural component* is a Transformer, while the *symbolic component* is a deterministic executor that applies the actions predicted by the Transformer to construct the state for the next reasoning or calculation step. The training data are derived from CoT traces explicitly providing the intermediate calculation steps.

In each CoT step of a trace, the tokens involved in the current calculation are tagged by a small set of positional tags. The neural model primarily predicts two kinds of information: (1) the local result of the current calculation step and (2) the state-edit actions, including the movements of the tags, required to construct the next step, and some control information.

Importantly, these predictions are made only from the tagged positions and their immediate neighbors, together with the control fields used by the task. We refer to this model-visible information as the *local context*. The Transformer is restricted by the attention mask to this bounded local information when producing the prediction. The resulting predictions are represented as a fixed-dimensional *action vector*, whose fields specify the effects that the symbolic executor applies to the current symbolic state to produce the next step. For example, consider the Addition problem $345 + 678$. The CoT data at the first step is as follows,

```
345 + 678 = ______        Carry = 0
  I     J      K
```

The tags I and J mark the digits 5 and 8, respectively, while K marks the blank cell or position at which the output digit will be written. The blank cells represent a pre-allocated result workspace, which need not be completely filled by the result. The symbolic state is represented by two aligned channels: a content channel and a positional-tag channel. Blank content cells and the absence of a positional tag are represented explicitly.

For this step, the Transformer predicts the local result of $5 + 8 + 0 = 13$: the output digit 3 and the updated carry 1. It also predicts that the three tags should move one position to the left for the next calculation step. The task-relevant part of the predicted action vector is therefore

$$(3, 1, \text{LEFT}, \text{LEFT}, \text{LEFT}),$$

where 3 is the predicted output digit, 1 is the predicted next carry, and the three LEFT fields specify the predicted movements of I, J, and K, respectively. The external symbolic executor applies these

predictions literally: it writes the digit, replaces the previous carry 0 with the new carry 1, and moves the tags as specified, producing the next symbolic state,

```
345 + 678 = _____3        Carry = 1
 I     J        K
```

This resulting state is then used for the next policy invocation. For simplicity of notation, we will use 4_I, 7_J and $_{}_K$ to represent the tagged positions above.

A key characteristic of our method is that the Transformer has access only to local contexts centered on the positional tags, together with a fixed-size set of control information, both of which are independent of the overall input length. Increasing the input length therefore requires only more applications of the same learned local policy, without increasing the information processed at each step. The Transformer predicts only a fixed-size action vector, while the symbolic executor deterministically materializes the predicted actions to construct the next state. This separation between bounded local neural prediction and deterministic symbolic state materialization provides the foundation for length generalization in the class of reasoning problems considered in this work.

3.1 Problem Setup

Let $x^{(n)} = (x_1, \ldots, x_n)$ denote an input CoT trace of length n from an instance of a reasoning problem or task P. Without loss of generality, we use only three positional tags in the following presentation, as they suffice for all our considered problems. However, the proposed framework supports any fixed number of tags independent of the input length. Let the three tags be I, J, and K, and the locations or positions in the input sequence at step t be $\tau_t = (I_t, J_t, K_t)$. I and J mark the locations involved in the current transition. For example, in an Addition step, I and J mark the current operand digits, while K marks the position of the result to be written.

Many reasoning tasks also require some information to be carried between steps or states. Again, without loss of generality, we denote this as *state control information* s_t and in this work, we use $s_t = (m_t, \phi_t)$, where m_t is a Memory value and ϕ_t is an optional categorical Phase, but the proposed framework permits any number of elements in s_t. For example, for Addition, Memory represents the incoming *carry*; for Subtraction, it represents *borrow*. Phase represents the task-specific operation stage of the current reasoning step. Designated Phase values trigger the calling of child policies (like function calls) for *compositional tasks* like Multiplication and Division. Phase `A` invokes Addition during Multiplication, while Phase `S` invokes Subtraction during Division. When Memory is not used by a task, it remains at the fixed neutral value 0. For tasks that do not use task-specific control, Phase has the fixed neutral value `N`.

The input at step t is C_t and is the serialization or concatenation of Memory and Phase in the first two fixed positions, followed by the input sequence regions in their natural order. For example, the input of the first step of the above Addition ($345_I + 678_J = ______K$) is as follows,

$$C_1 = \underbrace{0}_{\text{Memory}} \underbrace{\texttt{N}}_{\text{neutral Phase}} \texttt{A:}345_I|\texttt{B:}678_J|\texttt{R:}___\ldots_K|$$

Each region is prefixed by its label (`A`, `B`, or `R`) followed by a colon, and adjacent regions are separated by `|`. Since the required length of the result region (`R`) is not known in advance, in our implementation we allocate a fixed workspace of 1024 cells, which is sufficient for all experimental tasks considered in this work. This fixed allocation is an implementation choice rather than a requirement of our theoretical analysis. The length-invariance result in Section 3.5 assumes only that sufficient symbolic workspace is available for each problem instance.

3.2 CoT Traces and Action Target Construction

For a reasoning task or problem, the user provides training data consisting of a collection of problem instances expressed in CoT steps or traces. Let the CoT trace of a training instance provided by the user be $(C_0, C_1, \ldots, C_T)$, where C_T is the terminal state reached by an action with $\mathrm{Stop} = 1$. For each parent-policy (not calling a child policy) transition from C_t to C_{t+1}, a deterministic differencer *Diff* compares the two states and automatically constructs the target action vector α_t,

$$\alpha_t = \textit{Diff}(C_t, C_{t+1}).$$

The differencer *Diff* is shared across all tasks and requires no task-specific implementation. The implementation of *Diff* is included in Appendix A and the *supplementary materials*.

A state C_t whose Phase invokes a child policy is handled by a *composition runtime* mechanism defined for the corresponding reasoning problem or task. Consequently, no action vector is generated or predicted for the transition from C_t to C_{t+1}. Instead, the composition runtime invokes the designated child policy and uses its output to construct the next state C_{t+1}.[1]

Conceptually, an action vector contains the next control state s_{t+1}, the tag-movement operations, and an indicator of whether the reasoning process terminates after the action is applied. Together, these components specify the information needed to construct the input for the next reasoning step, C_{t+1}. Without loss of generality, our current implementation uses the following action vector $\alpha_t \in \mathcal{A}$ at step t,[2]

$$\alpha_t = \left(\alpha_t^{\text{result}}, \alpha_t^{\text{memory}}, \alpha_t^{\text{I_move}}, \alpha_t^{\text{J_move}}, \alpha_t^{\text{K_move}}, \alpha_t^{\text{phase}}, \alpha_t^{\text{stop}}\right). \tag{1}$$

This schema of the action vector is also shared across all reported tasks. Let us see an example. Based on

$$C_1 = \texttt{0 N A:345}_I\texttt{|B:678}_J\texttt{|R:___..._}_K\texttt{|}, \text{and}$$
$$C_2 = \texttt{1 N A:34}_I\texttt{5|B:67}_J\texttt{8|R:___..._}_K\texttt{3|},$$

the differencer *Diff* produces the target action vector $\alpha_1 = (3, 1, \text{LEFT}, \text{LEFT}, \text{LEFT}, \text{N}, 0)$, where each `LEFT` indicates that the corresponding tag should move one position to the left. The last two values, `N` and 0, indicate that the next Phase remains neutral and that the reasoning process has not yet terminated, respectively. When the symbolic executor applies α_1 to C_1, it should reconstruct the next state C_2.

For training, we combine the CoT traces from all training instances to form a collection of input–action pairs (C, α) for training the Transformer, where each input C is paired with its corresponding action vector α. Since each pair represents an individual reasoning step, all (C, α) pairs can be treated independently during training.

3.3 Learning to Predict the Action Vector Based on Local Contexts

Tag-Centered Local Contexts: Let p_m denote the serialized position of a tag $m \in \{I, J, K\}$ in C. For a fixed radius r, the neighborhood around a serialized position p_m is

$$\mathcal{N}_r(p_m) = \{p_m - r, \ldots, p_m, \ldots, p_m + r\} \tag{2}$$

In our implementation, we use $r = 1$. Setting $r = 0$ is insufficient because the model would be unable to determine whether a tag has reached a region boundary. The tag positions are used by the attention masks to identify the corresponding local neighborhoods for the Transformer. Consequently, the same tag-centered local content and control state s_t yield the same form of model-observable local information regardless of their positions in the input sequence, even when the sequence is substantially longer than those encountered during training.

Let $\mathcal{O}$ be the model-observable space and $\mathcal{L}(C_t) \in \mathcal{O}$ be the model-observable local information in C_t, i.e., Memory, Phase, and $\mathcal{N}_r(p_m)$ of each tag position p_m. The Transformer learns a local action policy $F_\theta : \mathcal{O} \to \mathcal{A}$, where $\mathcal{A}$ is a fixed-dimensional action space. At step t, for the current input, $\widehat{C}_t$, the action vector $\widehat{a}_t$ is predicted with

$$\widehat{a}_t = F_\theta\left(\mathcal{L}(\widehat{C}_t)\right). \tag{3}$$

Loss function. Recall that the training data for the Transformer consists of a collection of pairs (C, α), where C is the input sequence and α is the action vector to be predicted from C. Let $\alpha = (\alpha^{(1)}, \ldots, \alpha^{(d)})$, where $\alpha^{(i)}$ denotes the i-th action field and d is the dimension of the action vector. Training minimizes the following objective:

$$\mathcal{J}(\theta) = \sum_{i=1}^{d} \lambda_i q_i(\alpha)\, \ell_i\left(f_{\theta,i}(\mathcal{L}(C)), \alpha^{(i)}\right), \tag{4}$$

[1] More details about Phase values can be found in Appendix C and their usage examples can be found in the Multiplication and Division CoT schemes in Appendix F.

[2] Details about the actions and their domains can be found in Appendix B.

where λ_i is the weight assigned to action field i, and $q_i(\alpha) \in 0, 1$ is a mask indicating whether field i is applicable to the current transition. The function $f_{\theta,i}$ denotes the prediction head for action field i, and ℓ_i denotes its corresponding loss. All action heads condition on the same model-observable local information. We use cross-entropy loss for all action fields.

3.4 Constructing the Next-Step Input with the Symbolic Executor

At inference time, the symbolic executor $\mathcal{E}$ applies the action vector $\widehat{\alpha}_t$ predicted by the Transformer from the input $\widehat{C}_t$ to produce the next step input $\widehat{C}_{t+1}$:

$$\widehat{C}_{t+1} = \mathcal{E}\left(\widehat{C}_t, \widehat{\alpha}_t\right). \tag{5}$$

The implementation of $\mathcal{E}$ is included in the *supplementary material*.

Phase-based child policy invocation. When the Phase field of $\widehat{C}_t$ indicates that a child policy should be invoked, a task-specific *composition runtime* mechanism performs the invocation and uses the returned result to construct the next state input $\widehat{C}_{t+1}$. For example, during Multiplication, Phase `A` invokes the trained Addition policy, whereas during Division, Phase `S` invokes the trained Subtraction policy. Note that, in this case, the transition from $\widehat{C}_t$ to $\widehat{C}_{t+1}$ is not performed through action prediction followed by the executor's application of the predicted actions $\widehat{a}_t$. Instead, it is carried out directly by the composition runtime module of the symbolic component. In our implementation, the composition runtime is integrated into the executor, which together form the symbolic component of the proposed system. Further details can be found in Appendices C and F.

3.5 Theorem: Length Invariance and Perfect Length Generalization

Theorem 3.1 *Let P be a problem in a class of problems $\mathcal{P}$ satisfying the following assumptions:*

1. ***Locality.*** *The local result and action vector α required to construct the next reasoning step depend only on a fixed-size observable local context $\mathcal{L}(C)$, consisting of the tags and their fixed-size neighborhoods, together with a finite symbolic state. Both the local context and action space are independent of the input length.*
2. ***Length-independent transition rule or policy.*** *For every reachable state whose Phase does not invoke a child policy, there exists a single function*

   $$F : \mathcal{L} \to \mathcal{A}$$

   that maps the observable local context to the correct action vector regardless of the overall input length.
3. ***Exact local prediction.*** *For every reachable state whose Phase does not invoke a child policy, the learned Transformer satisfies*

   $$F_\theta(\mathcal{L}) = F(\mathcal{L})$$

   for every local context $\mathcal{L}$ that can arise from problem instances in P.
4. ***Correct symbolic execution.*** *The symbolic module correctly executes every predicted action, calls the child policy, and produces the next step input state.*

Then, for every problem instance $x \in P$ of arbitrary length, the resulting neuro-symbolic system produces the same sequence of intermediate CoT steps or states and the same final output as that of the underlying reasoning algorithm. Consequently, the system achieves perfect LG for P.

The proof is provided in Appendix D. In the next section, we empirically demonstrate that the proposed method achieves perfect LG across all evaluated tasks and input lengths.

4 Empirical Evaluation

We evaluate ten tasks: Copy, Reverse, ReCopy, Contains-101, Gray Code, Parity, Addition, Subtraction, Multiplication, and Division. Their detailed descriptions are given in Table 8 in Appendix E. An example CoT scheme used for training for each task is provided in Appendix F.

Training Data Generation. For the first six single-input tasks, training sequence lengths L range from 1 to 8. For the last four two-operand tasks, each operand contains between 1 and 8 digits.

For each single-input task, we randomly generate 100 instances at each training length, yielding $8 \times 100 = 800$ traces per task. For the two-operand arithmetic tasks, sampling is stratified by operand-width pairs and covers all ordered pairs in $\{1, \ldots, 8\} \times \{1, \ldots, 8\}$. For each width pair, the two operands are randomly sampled according to their specified widths. For example, for the width pair $(5, 3)$, the first operand is sampled from $[10000, 99999]$ and the second from $[100, 999]$.

For Addition and Subtraction, we generated 50 instances per width pair, yielding $8 \times 8 \times 50 = 3{,}200$ traces. For Multiplication and Division, we generate 100 instances per width pair, yielding 6,400 traces per task. Additionally, for Division, the divisor is constrained to be nonzero, but exact divisibility is not required; the task produces the integer quotient while maintaining the intermediate remainder throughout the reasoning process.

Test Instance Generation. Our experiments use six input lengths, $L \in \{8, 16, 32, 64, 100, 200\}$. For each single-input task, we generate 300 test instances for every value of L. For two-operand tasks, L denotes the maximum operand width (or length). To evaluate both balanced and asymmetric inputs, we use the width pairs (L, L), $(1, L)$, $(L, 1)$, $(\lfloor L/2 \rfloor, L)$, and $(L, \lfloor L/2 \rfloor)$. Operand values are generated randomly according to their selected widths as above. For each width pair, we generate 60 instances, which gives the total of 300 instances.

Implementation Details. Each learned policy uses the same 136,970-parameter Transformer architecture and is trained once from model seed 0. Multiplication and Division sequentially invoke a frozen Addition or Subtraction policy, respectively. The input is represented by two aligned channels. The content channel contains the original token sequence (i.e., C), and the tag channel indicates whether each corresponding cell carries tag I, J, or K. For example,

Content	A	:	3	4	5
Tag	∅	∅	∅	I	∅

Thus, the content token 4 and tag I are represented in separate channels but aligned at the same position. The two channels are embedded separately, concatenated, and projected to obtain the input representation. The attention mask is constructed from the tag channel. Under radius-one attention ($r = 1$), each tag query attends only to its tagged position, the immediately adjacent positions, and the Memory and Phase control entries. In the example above, the query for I observes the local content 3, 4, and 5. The same rule is applied to J and K. No absolute position indices are provided to the model. The Transformer produces one hidden representation for each of the three tag queries I, J, and K. We concatenate these three representations and pass the resulting vector to separate output heads. These heads predict the symbol to be written at the position or cell of the output tag, the next Memory value, the movement of each tag, the Phase, and whether the reasoning process should Stop. Complete architecture and optimization details are provided in Appendix G.

Baselines. We do not include comparisons with existing baselines because their performance is substantially worse. To the best of our knowledge, the strongest empirical results to date are reported in (Xiao & Liu, 2025). However, for Addition, their method achieves only 90% accuracy at $L \in [1, 41)$, with performance deteriorating substantially as L increases further. For Multiplication and Division, the accuracy begins to decline rapidly beyond $L = 16$, from about 98% at $L = 16$.

Evaluation Metric. We use exact task accuracy. A rollout is counted as correct only if the model predicts Stop and the final result exactly matches the ground-truth result. For Subtraction, the terminal Memory value must also correctly indicate whether underflow occurred. Positional tags need not be removed before termination.

4.1 Main Results

Table 1 reports the results of our method. As we can see, with the training length of $L = 8$, the proposed method achieves 100% accuracy (1.000) for all test lengths.

Stress Test. We further stress-test the proposed method at substantially longer lengths, using $L \in \{400, 600, 800, 1000\}$ for Copy, Parity, and Addition. Note that for Addition in this set of experiments, we use the same length for both operands. The proposed method achieves 1.000 accuracy across all tested lengths and tasks. We do not include Multiplication or Division in this

stress test because their evaluation at such lengths is computationally very expensive, and operands exceeding 200 digits are unlikely to arise in practical applications. Table 2 reports perfect results.

Table 1: Accuracy results of our method on 300 test instances for each task at each length L setting.

Task	$L = 8$	$L = 16$	$L = 32$	$L = 64$	$L = 100$	$L = 200$
Copy	1.000	1.000	1.000	1.000	1.000	1.000
Reverse	1.000	1.000	1.000	1.000	1.000	1.000
ReCopy	1.000	1.000	1.000	1.000	1.000	1.000
Contains-101	1.000	1.000	1.000	1.000	1.000	1.000
Gray Code	1.000	1.000	1.000	1.000	1.000	1.000
Parity	1.000	1.000	1.000	1.000	1.000	1.000
Addition	1.000	1.000	1.000	1.000	1.000	1.000
Subtraction	1.000	1.000	1.000	1.000	1.000	1.000
Multiplication	1.000	1.000	1.000	1.000	1.000	1.000
Division	1.000	1.000	1.000	1.000	1.000	1.000

Table 2: Accuracy of the long-length stress tests using 300 test instances for each L setting.

Task	$L = 400$	$L = 600$	$L = 800$	$L = 1000$
Copy	1.000	1.000	1.000	1.000
Parity	1.000	1.000	1.000	1.000
Addition	1.000	1.000	1.000	1.000

4.2 Ablation Experiments

We conduct two controlled comparisons using Copy and Addition, followed by a diagnostic experiment on a new task called Middle.

(1). Predicting the full successor state. Our proposed method predicts an action vector that is applied by the executor to construct the next state. In this variant, the model does not predict the action vector or use the executor, but directly predicts the successor state C_{t+1} from C_t.

We evaluate this variant under two attention mechanisms: local attention, which applies our locality restriction, and global attention, under which each output query attends to the complete state C_t. The successor-state models retain the backbone dimensions used by our model, but replace the action heads with the shared content- and marker-prediction heads. All models are trained only on lengths $L \in \{1, \ldots, 8\}$, as before. We additionally evaluate a higher-capacity global-attention model with 2,578,534 parameters on Addition to examine whether the results are explained by insufficient model capacity. The results are reported in Table 3.

Table 3: Accuracy under direct successor-state prediction. Each entry aggregates 300 test rollouts.

Task	Prediction target	Condition	$L = 8$	$L = 16$	$L = 32$	$L = 64$	$L = 100$	$L = 200$
Copy	Successor state	Local	1.000	0.000	0.000	0.000	0.000	0.000
Copy	Successor state	Global	1.000	0.000	0.000	0.000	0.000	0.000
Copy	Action vector (ours)	Radius-one local ($r = 1$)	1.000	1.000	1.000	1.000	1.000	1.000
Addition	Successor state	Local	0.110	0.080	0.047	0.000	0.000	0.000
Addition	Successor state	Global	0.000	0.000	0.000	0.000	0.000	0.000
Addition	Successor state	Large global	0.990	0.000	0.000	0.000	0.000	0.000
Addition	Action vector (ours)	Radius-one local ($r = 1$)	1.000	1.000	1.000	1.000	1.000	1.000

Both Copy successor-state models achieve perfect accuracy at the maximum training length $L = 8$ but fail at every longer length. For Addition, the standard local model retains partial accuracy at short extrapolation lengths but deteriorates as length increases. The standard global model performs poorly at $L = 8$, whereas the higher-capacity global model reaches 0.990 accuracy, nearly fitting the training-length distribution. However, it obtains zero accuracy at every evaluated length beyond the training range. These results demonstrate the advantage of our proposed neuro-symbolic approach that predicts action vectors.

(2). Observation scope. Here we experiment with $r = 0$, where each tag query observes only the position identified by that tag. Our method uses $r = 1$, which additionally includes the immediate left and right neighboring positions. Under global attention, each tag query attends to the complete state C_t. This comparison therefore tests how observation scope affects action-vector prediction.

As we can observe from Table 4, $r = 0$ performs poorly because this variant is not able to determine whether a token is a boundary one or an internal one.

Table 4: Accuracy under different observation scopes. Each entry aggregates 300 test rollouts.

Task	Observation scope	$L = 8$	$L = 16$	$L = 32$	$L = 64$	$L = 100$	$L = 200$
Copy	Exact tag ($r = 0$)	0.000	0.000	0.000	0.000	0.000	0.000
Copy	Radius-one local ($r = 1$)	1.000	1.000	1.000	1.000	1.000	1.000
Copy	Global	1.000	0.023	0.000	0.000	0.000	0.000
Addition	Exact tag ($r = 0$)	0.140	0.113	0.120	0.107	0.123	0.120
Addition	Radius-one local ($r = 1$)	1.000	1.000	1.000	1.000	1.000	1.000
Addition	Global	0.363	0.000	0.000	0.000	0.000	0.000

(3). Using a task whose output depends on input length. We introduce a simple controlled Middle diagnostic task to examine the limits of bounded local observation. The task is as follows: given a randomly generated binary sequence $(x_1, \ldots, x_n)$, the task is to output its middle token $x_{\lceil n/2 \rceil}$, using the left-middle position when n is even. Tag I is initially placed at the rightmost position of the sequence, and the policy must predict the middle token directly in its first action. The policy receives neither the input length nor the middle position, and the target token generally lies outside its bounded local observation. The lengths of the training sequences are in $[1, 8]$.

Table 5 reports the test results. The consistently low and somewhat random accuracy values show that the required output cannot generally be identified from the observation supplied to the policy. We can thus conclude that the required output cannot be determined from the supplied model-observable tag context when its position depends on the unobserved input length.

Table 5: Exact accuracy on the Middle diagnostic task. Each entry aggregates 300 test rollouts.

Task	$L = 8$	$L = 16$	$L = 32$	$L = 64$	$L = 100$	$L = 200$
Middle	0.430	0.483	0.540	0.463	0.507	0.497

5 Conclusion

This paper proposed a neuro-symbolic framework for length generalization (LG) in reasoning problems. The key idea is to make the prediction performed by the neural model (which is the Transformer in our case) at each reasoning step independent of the overall input length. Rather than predicting the full intermediate output at each step, whose size may grow with the input or output length, our neural model predicts a fixed-size action vector. Furthermore, attention masks restrict the Transformer to bounded local contexts centered on a fixed set of positional tags, together with fixed-size control information. The predicted actions are then deterministically materialized by a symbolic executor to construct the input for the next reasoning step. Consequently, increasing the input length requires only additional applications of the same learned local policy, rather than generalizing the neural computation itself to longer contexts or outputs. We evaluated the proposed framework on a diverse set of symbolic and arithmetic reasoning tasks. Despite being trained only on short instances, the method achieves perfect LG across all evaluated tasks and test lengths, including lengths substantially beyond those seen during training. Additional stress tests on selected tasks further demonstrate perfect accuracy at lengths up to $L = 1000$. These results provide empirical support for the central principle underlying our approach: when each reasoning transition can be expressed through bounded, length-independent local information and a fixed-size action space, reasoning over longer inputs can be reduced to repeatedly applying the same learned local transition.

Limitations

The main limitation of the proposed method is that it applies only to a class of reasoning problems for which the information required at each reasoning step can be localized and each action field can be predicted from a bounded local context that is independent of the overall input length. Thus the proposed approach does not provide a general solution to LG for all reasoning problems. For example, the approach cannot solve the Middle task above.

The second limitation is that the current differencer *Diff* and the executor $\mathcal{E}$ are designed and implemented specifically for the tasks considered in our experiments. For reasoning tasks that require actions not captured by the current action vector, the action space may need to be expanded, with *Diff* and $\mathcal{E}$ adapted accordingly, including the composition runtime module if the tasks need to call child policies. However, such expansions do not alter the proposed overall framework.

AI USE STATEMENT

Generative AI tools were used to assist with code review, debugging, reference discovery, and language polishing. The manuscript itself was written by the authors.

ETHICS STATEMENT

This work uses synthetic symbolic and arithmetic tasks with generated data and involves no personal, human-subject, or sensitive information. We therefore do not identify any ethical risks associated with this work.

REPRODUCIBILITY STATEMENT

The code of this work has been submitted in supplementary materials.

REFERENCES

Emmanuel Abbe, Samy Bengio, Aryo Lotfi, Colin Sandon, and Omid Saremi. How far can transformers reason? the globality barrier and inductive scratchpad. *Advances in Neural Information Processing Systems*, 37:27850–27895, 2024.

Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://openreview.net/forum?id=zSkYVeX7bC4.

Pranjal Awasthi, Anupam Gupta, and Ravi Kumar. Length generalization via auxiliary tasks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. Training neural networks as recognizers of formal languages. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ta-Chung Chi, Ting-Han Fan, Alexander I. Rudnicky, and Peter J. Ramadge. Transformer working memory enables regular language reasoning and natural language length extrapolation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5972–5984, 2023.

Hanseul Cho, Jaeyoung Cha, Pranjal Awasthi, Srinadh Bhojanapalli, Anupam Gupta, and Chulhee Yun. Position coupling: Improving length generalization of arithmetic transformers using task structure. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/27aa3a0e6d63db269977bb2df5607cb8-Abstract-Conference.html.

Hanseul Cho, Jaeyoung Cha, Srinadh Bhojanapalli, and Chulhee Yun. Arithmetic transformers can length-generalize in both operand length and count. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eIgGesYKLG.

Jishnu Ray Chowdhury and Cornelia Caragea. Monotonic location attention for length generalization. *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, 2023a.

Jishnu Ray Chowdhury and Cornelia Caragea. Recursion in recursion: Two-level nested recursion for length generalization with scalability. *Proceedings of 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023b.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A Ortega. Neural networks and the Chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*, 2023.

Shaoxiong Duan, Yining Shi, and Wei Xu. From interpolation to extrapolation: Complete length generalization for arithmetic transformers. *arXiv preprint arXiv:2310.11984*, 2023.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/deb3c28192f979302c157cb653c15e90-Abstract.html.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3991–4008. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.222. URL https://aclanthology.org/2024.naacl-long.222/.

Zhenyu He, Guhao Feng, Shengjie Luo, Kai Yang, Liwei Wang, Jingjing Xu, Zhi Zhang, Hongxia Yang, and Di He. Two stones hit one bird: Bilevel positional encoding for better length extrapolation. *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. Case-based or rule-based: How do transformers do the math? *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Zhouqi Hua, Wenwei Zhang, Chengqi Lyu, Yuzhe Gu, Songyang Gao, Kuikun Liu, Dahua Lin, and Kai Chen. The imitation game: Turing machine imitator is length generalizable reasoner. In *International Conference on Learning Representations (ICLR 2026)*, 2026.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. A formal framework for understanding length generalization in transformers. *ICLR*, 2025.

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

Samy Jelassi, Stéphane d'Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. In *Advances in Neural Information Processing Systems*, volume 36, pp. 24892–24928, 2023.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations*, 2024.

Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional interpolation for relative positions improves long context transformers. In *The Twelfth International Conference on Learning Representations*, 2024.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, et al. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37: 108012–108041, 2024.

Anh T Nguyen, Saleh Momeni, Ashutosh Chaubey, Changnan Xiao, and Bing Liu. Push, pop, parallelize: Stack-augmented linear attention via the delta rule. In *Proceedings of the Forty-third International Conference on Machine Learning (ICML-2026)*, 2026.

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. URL https://arxiv.org/abs/2112.00114.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1889–1903. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-short.161. URL https://aclanthology.org/2023.acl-short.161/.

Mahdi Sabbaghi, George Pappas, Hamed Hassani, and Surbhi Goel. Explicitly encoding structural symmetry is key to length generalization in arithmetic tasks. *arXiv preprint arXiv:2406.01895*, 2024. URL https://arxiv.org/abs/2406.01895.

Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. A length-extrapolatable transformer. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021.

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

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. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Noam Wies, Yoav Levine, and Amnon Shashua. Sub-task decomposition enables learning in sequence to sequence tasks. *Proceedings of International Conference on Learning Representations (ICLR-2023)*, 2023.

Changnan Xiao and Bing Liu. Generalizing reasoning problems to longer lengths. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zpENPcQSj1.

Liang Zhao, Xiachong Feng, Xiaocheng Feng, Weihong Zhong, Dongliang Xu, Qing Yang, Hongtao Liu, Bing Qin, and Ting Liu. Length extrapolation of transformers: A survey from the perspective of positional encoding. *Findings of EMNLP-2024*, 2024.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *Proceedings of International Conference on Learning Representations (ICLR-2024)*, 2024a.

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

APPENDIX

A DIFFERENCER AND ROUND-TRIP VALIDATION

For each model-predicted transition (C_t, C_{t+1}), the shared differencer derives $\alpha_t = \mathit{Diff}(C_t, C_{t+1})$. It accepts at most one changed content cell or position, which must be identified by tag K. Tag displacements must be KEEP, LEFT, or RIGHT; a tag may become absent only through an outward movement at its region boundary. The derived action is retained only if applying it to C_t reconstructs C_{t+1} exactly. Transitions beginning in invocation Phase `A` or `S` are excluded because they are handled by the composition runtime rather than converted into parent-policy action targets. The code of the differencer is provided in the *supplementary materials*.

B ACTIONS AND THEIR DOMAINS

The action vector used by all reported tasks is

$$\alpha_t = \left(\alpha_t^{\text{result}}, \alpha_t^{\text{memory}}, \alpha_t^{I\text{-move}}, \alpha_t^{J\text{-move}}, \alpha_t^{K\text{-move}}, \alpha_t^{\text{phase}}, \alpha_t^{\text{stop}}\right). \tag{6}$$

Table 6 lists the domain and execution effect of each field.

Table 6: Action fields description.

Field	Domain	Execution effect
Result (output token)	$\mathcal{V}$, including BLANK	Predicted token $\mathcal{V}$ to be written at the cell or position identified by tag K; BLANK produces no write.
Memory	$\{0, \ldots, 9\}$	Predicted Memory value.
I-move	{KEEP, LEFT, RIGHT}	Predicted relative movement to tag I. An outward movement at the boundary of its current region makes the tag absent.
J-move	{KEEP, LEFT, RIGHT}	Predicted relative movement to tag J. An outward movement at the boundary of its current region makes the tag absent.
K-move	{KEEP, LEFT, RIGHT}	Predicted relative movement to tag K. An outward movement at the boundary of its current region makes the tag absent.
Phase	Task-specific categorical domain (details below)	Predicted task-control state and, for designated values, invokes a trained child policy.
Stop	$\{0, 1\}$	Terminates the reasoning process when its value is 1.

A nonblank Result token modifies only the content cell or position currently identified by K, whereas BLANK is the designated no-write value and leaves that cell unchanged. All other content cells are preserved by the symbolic executor. Tag movements are relative and never contain an absolute position. When a predicted movement points outward at the locally visible boundary of the tag's current region, that tag becomes inactive rather than entering an adjacent region. If a tag is already absent, its movement field is excluded from the training loss and ignored by the executor.

The Stop field is predicted by the policy from the same tag-local observation, Memory, and Phase used by the other action heads. Region boundaries visible in the local neighborhoods indicate whether more symbols remain, while Memory and Phase distinguish task-specific conditions. When $\text{Stop} = 1$, the executor applies the other predicted action fields and then terminates the procedure; the positional tags need not first become inactive.

C PHASE DOMAIN

Phase represents the task-specific control stage of the current reasoning step. The current Phase is included in the model input, and the next Phase is predicted as part of the action vector. Tasks that do not require task-specific control use only the neutral value `N`. Multiplication and Division use the Phase domains shown in Table 7.

Table 7: Task-specific Phase values and their operational meanings.

Task	Phase	Meaning
Tasks without phase-dependent control	N	Remains at the fixed neutral value.
Multiplication	M	Constructs the current shifted partial product in R, using Memory for the multiplication carry.
Multiplication	A	Invokes the trained Addition child policy on the partial product in R and the externally stored SavedResult.
Multiplication	X	Handles the remaining parent-side operations, including a residual carry, tag repositioning, and termination.
Division	S	Invokes the trained Subtraction child policy on the active dividend prefix ending at tag I and the divisor in B.
Division	D	Processes the Subtraction return and predicts the next quotient, tag-movement, invocation, or termination action.

The parent policy is evaluated only in non-invocation (no function call) Phases: M and X for Multiplication, and D for Division. In these Phases, the Transformer predicts the action fields, which the symbolic executor applies literally.

In child policy (or function) invocation Phases A and S, the composition runtime constructs the child policy's Step 1 input from the task-declared operand specification and runs the frozen child policy until it predicts Stop. It then applies the child output and terminal control information according to the return specification and resumes the parent in the designated continuation Phase. The runtime performs no arithmetic or numerical comparison.

Why Multiplication requires Phase X. Phase M is reserved for partial-product digit construction. Phase X handles the remaining parent-side operations in Multiplication, such as writing a remaining carry, resetting Memory, repositioning the operand and result tags for the next multiplier digit, and predicting Stop after the final Addition return. Using Phase M for both partial-product construction and these other parent-side operations could associate locally similar states with different required actions under the same control value. Phase X separates these roles without introducing a hidden operation; while X is active, the model still predicts every Result, Memory, movement, Phase, and Stop field, i.e., the action vector. See an example in Appendix F.

D Proof of Theorem 3.1

Let $C_1, C_2, \ldots, C_T$ denote the sequence of states produced by the underlying algorithm, and let $\widehat{C}_1, \widehat{C}_2, \ldots, \widehat{C}_T$ denote the states produced by the neuro-symbolic system.

We prove by induction that $\widehat{C}_t = C_t$ for every calculation or reasoning step t.

Base case. The initial state is constructed identically by the algorithm and the neuro-symbolic system, so

$$\widehat{C}_1 = C_1.$$

Therefore, the tag positions, symbolic state, and local context are identical:

$$\mathcal{L}(\widehat{C}_1) = \mathcal{L}(C_1).$$

By the exact local prediction assumption,

$$F_\theta(\mathcal{L}(\widehat{C}_1)) = F(\mathcal{L}(C_1)).$$

The symbolic module correctly executes this action, yielding

$$\widehat{C}_2 = C_2$$

Inductive step. Assume that

$$\widehat{C}_t = C_t.$$

Then the tag positions, local neighborhoods, and symbolic state are identical. By locality,

$$\mathcal{L}(\widehat{C}_t) = \mathcal{L}(C_t).$$

The Transformer therefore produces

$$F_\theta(\mathcal{L}(\widehat{C}_t)) = F(\mathcal{L}(C_t)).$$

By the correctness of symbolic execution,

$$\begin{aligned}\widehat{C}_{t+1} &= \mathcal{E}\left(\widehat{C}_t, F_\theta(\mathcal{L}(\widehat{C}_t))\right) \\ &= \mathcal{E}\left(C_t, F(\mathcal{L}(C_t))\right) \\ &= C_{t+1}.\end{aligned}$$

Thus,

$$\widehat{C}_{t+1} = C_{t+1}.$$

By induction,

$$\widehat{C}_t = C_t$$

for every calculation step t. Therefore, the final outputs are identical:

$$\widehat{y}(x) = y(x).$$

Since none of the assumptions depends on the input length n, the same result holds for arbitrary input lengths. Hence, the neuro-symbolic system achieves perfect length generalization for the problem P in class $\mathcal{P}$. □

Note that, for simplicity, the above proof does not explicitly consider states in which the composition runtime module invokes a child policy. Such states can be readily incorporated into the proof, since we assume that the corresponding composition runtime module correctly uses the output of the child policy to construct the next state.

E Task Definitions

Table 8 describes each reasoning task used in our experiments.

Table 8: Tasks used in the main evaluation.

Task	Definition
Copy	Reproduce the input sequence.
Reverse	Produce the input sequence in reverse order.
ReCopy	Process the input digits from left to right while preserving their order. Each digit in $\{1, 2, 3\}$ is copied once, each digit in $\{4, 5, 6\}$ is copied three times, and each digit in $\{7, 8, 9\}$ is copied five times. For example, the input 147 produces 144477777.
Contains-101	Given a binary sequence, output 1 if it contains the substring `101`, and output 0 otherwise.
Gray Code	Given a binary sequence $x_1, \ldots, x_n$, produce the sequence $y_1, \ldots, y_n$, where $y_1 = x_1$ and $y_i = x_{i-1} \oplus x_i$ for every $i > 1$. For example, the input 1011 produces 1110.
Parity	Given a binary sequence, output 1 if it contains an odd number of 1's and output 0 otherwise.
Addition	Produce the decimal sum of two nonnegative integers.
Subtraction	Given two decimal integers with maximum width w, write $(A-B) \bmod 10^w$ as a width-w result in R. The terminal borrow stored in Memory indicates whether $A < B$.
Multiplication	Given two nonnegative decimal integers, produce their product.
Division	Given a decimal dividend and a nonzero variable-width divisor, produce the integer quotient, zero-padded to the width of the dividend. Exact divisibility is not required.

F Task-Level CoT Scheme Examples

The following examples illustrate representative state transitions for each task. For the more involved compositional tasks, Multiplication and Division, we provide complete parent-state traces.

Copy. The raw input is $A = \texttt{t2V}$. The first three states are

$$C_1 : \texttt{0 N} \quad \texttt{A:t}_I\texttt{2V|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_2 : \texttt{0 N} \quad \texttt{A:t2}_I\texttt{V|} \quad \texttt{R:t}_{K}\texttt{...|},$$
$$C_3 : \texttt{0 N} \quad \texttt{A:t2V}_I\texttt{|} \quad \texttt{R:t2}_{K}\texttt{...|}.$$

The corresponding actions are

$$\alpha_1 = (\texttt{t}, 0, \textsc{Right}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0),$$
$$\alpha_2 = (\texttt{2}, 0, \textsc{Right}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0).$$

Each action copies the symbol identified by I to the cell or position identified by K, then advances both tags. After the final symbol is written, the terminal action predicts $\mathrm{Stop} = 1$ without removing the tags.

Reverse. The raw input is $A = \texttt{Nxp}$. The first three states are

$$C_1 : \texttt{0 N} \quad \texttt{A:Nxp}_I\texttt{|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_2 : \texttt{0 N} \quad \texttt{A:Nx}_I\texttt{p|} \quad \texttt{R:p}_{K}\texttt{...|},$$
$$C_3 : \texttt{0 N} \quad \texttt{A:N}_I\texttt{xp|} \quad \texttt{R:px}_{K}\texttt{...|}.$$

The corresponding actions are

$$\alpha_1 = (\texttt{p}, 0, \textsc{Left}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0),$$
$$\alpha_2 = (\texttt{x}, 0, \textsc{Left}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0).$$

Thus, I scans the input from right to left while K advances from left to right through the result workspace.

ReCopy. The raw input is $A = \texttt{958}$. The first three states are

$$C_1 : \texttt{0 N} \quad \texttt{A:9}_I\texttt{58|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_2 : \texttt{5 N} \quad \texttt{A:9}_I\texttt{58|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_3 : \texttt{4 N} \quad \texttt{A:9}_I\texttt{58|} \quad \texttt{R:9}_{K}\texttt{...|}.$$

The corresponding actions are

$$\alpha_1 = (\textsc{Blank}, 5, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 0),$$
$$\alpha_2 = (\texttt{9}, 4, \textsc{Keep}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0).$$

The first action initializes Memory with the repetition count for `9`. The second writes the first copy and decrements Memory. The policy continues writing `9` until Memory reaches 0, after which it advances I to the next input digit.

Contains-101. Consider the raw input $A = \texttt{101}$. The first three states are

$$C_1 : \texttt{0 N} \quad \texttt{A:1}_I\texttt{01|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_2 : \texttt{1 N} \quad \texttt{A:10}_I\texttt{1|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_3 : \texttt{2 N} \quad \texttt{A:101}_I\texttt{|} \quad \texttt{R:}_{K}\texttt{...|}.$$

The corresponding actions are

$$\alpha_1 = (\textsc{Blank}, 1, \textsc{Right}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 0),$$
$$\alpha_2 = (\textsc{Blank}, 2, \textsc{Right}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 0).$$

Memory records progress through the target pattern. No result is written while the scan is incomplete. After the final input bit completes `101`, the terminal action writes `1` and predicts $\mathrm{Stop} = 1$.

Gray Code. The raw input is $A = \texttt{010}$. The first three states are

$$C_1 : \texttt{0 N} \quad \texttt{A:0}_I\texttt{1}_J\texttt{0|} \quad \texttt{R:}_{K}\texttt{...|},$$
$$C_2 : \texttt{0 N} \quad \texttt{A:0}_I\texttt{1}_J\texttt{0|} \quad \texttt{R:0}_{K}\texttt{...|},$$
$$C_3 : \texttt{0 N} \quad \texttt{A:01}_I\texttt{0}_J\texttt{|} \quad \texttt{R:01}_{K}\texttt{...|}.$$

The corresponding actions are

$$\alpha_1 = (0, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Right}, \texttt{N}, 0),$$
$$\alpha_2 = (1, 0, \textsc{Right}, \textsc{Right}, \textsc{Right}, \texttt{N}, 0).$$

The first action writes $y_1 = x_1$. The second writes $x_1 \oplus x_2$ and advances all three tags to the next pair and output position.

Parity. The raw input is $A = \texttt{101}$. The first three states are

$$\begin{aligned} C_1 &: \texttt{0 N}\ \ \texttt{A:1}_I\texttt{01|}\ \ \texttt{R:}_{K}\texttt{...|}, \\ C_2 &: \texttt{1 N}\ \ \texttt{A:10}_I\texttt{1|}\ \ \texttt{R:}_{K}\texttt{...|}, \\ C_3 &: \texttt{1 N}\ \ \texttt{A:101}_I\texttt{|}\ \ \texttt{R:}_{K}\texttt{...|}. \end{aligned}$$

The corresponding actions are

$$\begin{aligned} \alpha_1 &= (\textsc{Blank}, 1, \textsc{Right}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 0), \\ \alpha_2 &= (\textsc{Blank}, 1, \textsc{Right}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 0). \end{aligned}$$

Memory stores the running parity. The first bit changes it from 0 to 1, whereas the second bit leaves it unchanged. The result is written only by the terminal action after the complete input has been processed.

Addition. The raw input is $A = \texttt{24}$ and $B = \texttt{17}$. The complete two-transition trace is

$$\begin{aligned} C_1 &: \texttt{0 N}\ \ \texttt{A:24}_I\texttt{|}\ \ \texttt{B:17}_J\texttt{|}\ \ \texttt{R:...}_{__K}\texttt{|}, \\ C_2 &: \texttt{1 N}\ \ \texttt{A:2}_I\texttt{4|}\ \ \texttt{B:1}_J\texttt{7|}\ \ \texttt{R:...}_{_K}\texttt{1|}, \\ C_3 &: \texttt{0 N}\ \ \texttt{A:2}_I\texttt{4|}\ \ \texttt{B:1}_J\texttt{7|}\ \ \texttt{R:...}_{_K}\texttt{41|}, \quad \mathrm{Stop} = 1. \end{aligned}$$

The corresponding actions are

$$\begin{aligned} \alpha_1 &= (\texttt{1}, 1, \textsc{Left}, \textsc{Left}, \textsc{Left}, \texttt{N}, 0), \\ \alpha_2 &= (\texttt{4}, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 1). \end{aligned}$$

The first action writes the least-significant digit of $4 + 7$, stores carry 1 in Memory, and moves the three tags left. The second action writes $2 + 1 + 1 = 4$, resets Memory to 0, and predicts $\mathrm{Stop} = 1$. Thus, carry consumption and termination are both model-predicted.

Subtraction. The raw input is $A = \texttt{29}$ and $B = \texttt{14}$. The complete two-transition trace is

$$\begin{aligned} C_1 &: \texttt{0 N}\ \ \texttt{A:29}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:...}_{_K}\texttt{|}, \\ C_2 &: \texttt{0 N}\ \ \texttt{A:2}_I\texttt{9|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:...}_{K}\texttt{5|}, \\ C_3 &: \texttt{0 N}\ \ \texttt{A:2}_I\texttt{9|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:...}_{K}\texttt{15|}, \quad \mathrm{Stop} = 1. \end{aligned}$$

The corresponding actions are

$$\begin{aligned} \alpha_1 &= (\texttt{5}, 0, \textsc{Left}, \textsc{Left}, \textsc{Left}, \texttt{N}, 0), \\ \alpha_2 &= (\texttt{1}, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{N}, 1). \end{aligned}$$

The first action writes $9 - 4 = 5$ and advances to the next column. The second writes $2 - 1 = 1$ and terminates directly. Memory remains 0 because neither column produces a borrow.

Multiplication. The raw input is $A = \texttt{21}$ and $B = \texttt{14}$. The complete sequence of parent states is shown below, with SavedResult included alongside each state for clarity:

$$\begin{aligned} C_1 &: \texttt{0 M}\ \ \texttt{A:21}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:}_{___K}\texttt{|}, && \mathrm{SavedResult} = 0, \\ C_2 &: \texttt{0 M}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:}_{__K}\texttt{4|}, && \mathrm{SavedResult} = 0, \\ C_3 &: \texttt{0 A}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:}_{_}\texttt{8}_K\texttt{4|}, && \mathrm{SavedResult} = 0, \\ C_4 &: \texttt{0 X}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:}_{__K}\texttt{_|}, && \mathrm{SavedResult} = 84, \\ C_5 &: \texttt{0 X}\ \ \texttt{A:21}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:}_{___K}\texttt{|}, && \mathrm{SavedResult} = 84, \\ C_6 &: \texttt{0 M}\ \ \texttt{A:21}_I\texttt{|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:}_{__K}\texttt{_|}, && \mathrm{SavedResult} = 84, \\ C_7 &: \texttt{0 M}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:}_{_K}\texttt{1_|}, && \mathrm{SavedResult} = 84, \\ C_8 &: \texttt{0 A}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:}_{}\texttt{2}_K\texttt{1_|}, && \mathrm{SavedResult} = 84, \\ C_9 &: \texttt{0 X}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:}_{K}\texttt{_|}, && \mathrm{SavedResult} = 294, \\ C_{10} &: \texttt{0 X}\ \ \texttt{A:2}_I\texttt{1|}\ \ \texttt{B:1}_J\texttt{4|}\ \ \texttt{R:}_{K}\texttt{_|}, && \mathrm{SavedResult} = 294, \quad \mathrm{Stop} = 1. \end{aligned}$$

The corresponding parent-policy actions are

$$\alpha_1 = (4, 0, \textsc{Left}, \textsc{Keep}, \textsc{Left}, \texttt{M}, 0),$$
$$\alpha_2 = (8, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{A}, 0),$$
$$\alpha_4 = (\textsc{Blank}, 0, \textsc{Right}, \textsc{Keep}, \textsc{Right}, \texttt{X}, 0),$$
$$\alpha_5 = (\textsc{Blank}, 0, \textsc{Keep}, \textsc{Left}, \textsc{Left}, \texttt{M}, 0),$$
$$\alpha_6 = (1, 0, \textsc{Left}, \textsc{Keep}, \textsc{Left}, \texttt{M}, 0),$$
$$\alpha_7 = (2, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{A}, 0),$$
$$\alpha_9 = (\textsc{Blank}, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{X}, 1).$$

These action vectors produce the transitions $C_1 \to C_2$, $C_2 \to C_3$, $C_4 \to C_5$, $C_5 \to C_6$, $C_6 \to C_7$, $C_7 \to C_8$, and $C_9 \to C_{10}$, respectively. There is no α_3 or α_8 because the transitions $C_3 \to C_4$ and $C_8 \to C_9$ begin in invocation Phase A. For these transitions, the composition runtime invokes the frozen Addition policy and constructs the resumed parent state from its return.

SavedResult is shown at each step for clarity. It is external runtime storage used only by Multiplication and is not part of the Transformer observation or the predicted action vector. Its initial value is 0, and it is absent from all other tasks. The composition runtime updates SavedResult with the results returned by successive invocations of the frozen Addition policy. Upon termination ($\mathrm{Stop} = 1$), it contains the final multiplication result.

The transition from C_1 to C_2 writes $1 \times 4 = 4$ and moves I and K one cell to the left. The next action writes $2 \times 4 = 8$, completing the first partial product 84, and predicts Phase A. Because C_3 has an invocation Phase, the parent Multiplication policy is not queried for $C_3 \to C_4$. The composition runtime invokes the frozen Addition policy on SavedResult 0 and partial product 84, stores the returned result 84 in SavedResult, blanks R, preserves the parent tag positions, and resumes Multiplication in Phase X.

The parent-policy transitions $C_4 \to C_5$ and $C_5 \to C_6$ reposition I, J, and K for the next multiplier digit and return to Phase M. The policy then writes the shifted partial product 210 in $C_6 \to C_8$ and again predicts Phase A. The runtime invokes Addition on SavedResult 84 and partial product 210, producing SavedResult 294 and the resumed state C_9. Finally, the Multiplication policy predicts $\mathrm{Stop} = 1$, producing C_{10}. No multiplication carry occurs in this example, so Memory remains zero throughout the trace.

Division. Division is solved using repeated application of Subtraction. Region A initially contains the dividend and is updated in place during long division: after each successful Subtraction invocation, the active dividend prefix is replaced by the subtraction result. Therefore, A contains the current partial remainder during execution and the final remainder at termination, while region R contains the quotient.

As an example, we use dividend $A = 28$ and divisor $B = 14$. The complete parent-state sequence is

$$\begin{aligned}
&C_1 : 0\ \texttt{S}\ \ \texttt{A:_2}_I\texttt{8|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:_}_K\texttt{_|},\\
&C_2 : 1\ \texttt{D}\ \ \texttt{A:_2}_I\texttt{8|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:_}_K\texttt{_|},\\
&C_3 : 0\ \texttt{S}\ \ \texttt{A:_28}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:0_}_K\texttt{|},\\
&C_4 : 0\ \texttt{D}\ \ \texttt{A:_14}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:0_}_K\texttt{|},\\
&C_5 : 0\ \texttt{S}\ \ \texttt{A:_14}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:01}_K\texttt{|},\\
&C_6 : 0\ \texttt{D}\ \ \texttt{A:__0}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:01}_K\texttt{|},\\
&C_7 : 0\ \texttt{S}\ \ \texttt{A:__0}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:02}_K\texttt{|},\\
&C_8 : 1\ \texttt{D}\ \ \texttt{A:__0}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:02}_K\texttt{|},\\
&C_9 : 1\ \texttt{D}\ \ \texttt{A:__0}_I\texttt{|}\ \ \texttt{B:14}_J\texttt{|}\ \ \texttt{R:02}_K\texttt{|},\quad \mathrm{Stop} = 1.
\end{aligned}$$

The corresponding parent-policy action vectors are

$$\alpha_2 = (0, 0, \textsc{Right}, \textsc{Keep}, \textsc{Right}, \texttt{S}, 0),$$
$$\alpha_4 = (1, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{S}, 0),$$
$$\alpha_6 = (2, 0, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{S}, 0),$$
$$\alpha_8 = (\textsc{Blank}, 1, \textsc{Keep}, \textsc{Keep}, \textsc{Keep}, \texttt{D}, 1).$$

These action vectors produce the transitions $C_2 \to C_3$, $C_4 \to C_5$, $C_6 \to C_7$, and $C_8 \to C_9$, respectively. Note that we don't have α_1, α_3, α_5 and α_7 because the transitions $C_1 \to C_2$, $C_3 \to C_4$, $C_5 \to C_6$, and $C_7 \to C_8$ begin in invocation Phase S. For these transitions, the composition runtime invokes the frozen Subtraction policy and constructs the next parent state from its return, without requesting an action prediction from the parent Division policy.

Phase S invokes the learned and frozen Subtraction policy. The composition runtime constructs the child input by reading the portion of A ending at tag I together with the divisor in B; it does not select a prefix length numerically. The position of I is controlled by actions predicted by the Division policy. At C_1, I is aligned with digit 2, so the runtime supplies 2 and 14 to Subtraction. The child underflows and returns terminal Memory value 1. The runtime transfers this value to the parent state, preserves the original prefix, and resumes Division in Phase D, producing C_2.

At C_2, the Division policy predicts an action that writes quotient digit 0, moves I and K right, resets Memory, and returns to Phase S. The symbolic executor applies this action, producing C_3. Moving I extends the active prefix from 2 to 28. Subtraction then returns 14 with terminal Memory value 0. The runtime transfers this Memory value, replaces the active prefix with 14, and resumes Division in Phase D, producing C_4.

The Division policy next updates the current quotient digit from 0 to 1 and invokes Subtraction again. The call on $14 - 14$ returns 0 with terminal Memory value 0, producing C_6. The policy then updates the same quotient digit from 1 to 2 and invokes Subtraction on $0 - 14$. This call underflows and returns terminal Memory value 1, so the runtime transfers this value, preserves the active prefix 0, and produces C_8. The Division policy finally predicts $\mathrm{Stop} = 1$, producing C_9 while retaining Memory value 1. The terminal state contains quotient 02 in R and remainder 0 in A.

Thus, the Division policy predicts quotient updates, tag movements, Phase transitions, and termination; the symbolic executor applies these actions; the frozen Subtraction policy performs subtraction and produces the terminal underflow information; and the composition runtime constructs the child inputs and transfers the declared return values.

G Implementation and Reproducibility

Architecture and tag representations. Every main policy uses the same 136,970-parameter Transformer and a shared 65-symbol vocabulary. The model has hidden width 48, four attention heads, and one relative-attention block. Content symbols and positional tags are embedded through separate channels, concatenated, and projected into a shared hidden space. The implementation names the result tag O; throughout the paper, we denote this tag by K.

For each positional tag $m \in \{I, J, K\}$, let $\widetilde{z}_m(C)$ denote its contextualized representation and let $h_m^{\mathrm{raw}}(C)$ denote the pre-attention representation of the content token carrying that tag. The representation supplied to the action heads is

$$z_m(C) = W_z \left[\widetilde{z}_m(C); h_m^{\mathrm{raw}}(C)\right], \qquad z(C) = [z_I(C); z_J(C); z_K(C)]. \tag{7}$$

The three tag representations are concatenated and passed through a shared adapter before reaching the separate action heads. If a positional tag is absent from the current state, its representation is set to a fixed zero vector.

Data and optimization. A separate policy is trained for each task using Adam with a learning rate of 10^{-4} and a minibatch size of 128. Each sampled transition is terminal with probability 0.1 when both terminal and non-terminal transitions are available. Copy, Reverse, ReCopy, Contains-101, Gray Code, Parity, Addition, and Subtraction are trained for 10,000 parameter updates. Division is trained for 20,000 updates, and Multiplication is trained for 60,000 updates. We did not use more

updates as these are sufficient. All reported main-task evaluations use the checkpoint from the final training update and independently generated test inputs. Training uses model seed 0; test results aggregate 100 inputs for each of three input-generation seeds, which give us 300 test instances in total.

For the successor-state ablation, the models based on our architecture are trained for 10,000 updates with learning rate 10^{-4}. The large global-attention Addition model is trained for 30,000 updates with learning rate 5×10^{-5}; the lower learning rate is used because preliminary runs exhibited unstable optimization dynamics.