Exploring Depth Generalization in Large Language Models for Solving Recursive Logic Tasks

Zhiyuan He
University College London
nick.he.21@ucl.ac.uk

Abstract

Large language models have demonstrated remarkable capabilities across many tasks, yet face significant challenges when dealing with recursive reasoning problems, those requiring the resolution of nested hierarchical structures. While prior research has extensively studied length generalization (a model's ability to handle longer sequences than seen during training), we investigate a distinct and underexplored limitation: depth generalization. Here, depth refers to the number of nested levels in a hierarchical problem, such as the layers of parentheses in a mathematical expression or the nesting of logical clauses in a Boolean formula. Our work reveals that standard transformer architectures struggle with problems involving deeper recursion than encountered during training, even when they perform well on longer but non-nested sequences. This limitation stems from their inability to maintain stack-like behavior, the capacity to track and resolve multiple levels of nested dependencies. Through systematic analysis, we demonstrate how this architectural constraint leads to rapid performance decay as the depth of the recursion increases. To address this challenge, we develop a novel looped locate-and-replace pipeline that decomposes recursive problems into manageable subcomponents. The approach employs two specialized models: a locator that identifies solvable subexpressions and a replacer that evaluates these components while preserving the overall structure. We evaluate this method in three carefully designed domains: Boolean algebra, recursive arithmetic, and propositional logic, each with a controllable depth of recursion. We show that our method effectively alleviates the performance decay when tested on out-of-distribution recursion depth.

1 Introduction

Large language models (LLMs), particularly transformer-based architectures (Vaswani et al. 2017), have achieved remarkable success across diverse domains, from natural language processing to symbolic reasoning (Brown et al. 2020; Radford et al. 2019). However, as their adoption grows, understanding their fundamental limitations becomes critical. Recent research has extensively explored the boundaries of transformers, with *length generalization* (Lin et al. 2025; Xiao and Liu 2025; Cai et al. 2025; Zhou et al. 2024; Abbe et al. 2024b,a; Li et al. 2024; Anil et al. 2022), the ability to generalize to longer sequences than those seen during training, being a prominent focus. Tasks such as multi-digit arithmetic, copying, and sorting have served as benchmarks for these studies, revealing both strengths and failures in extrapolating to longer inputs.

Yet, length is only one dimension of generalization. In this work, we investigate a distinct and underexplored axis: *depth generalization*, where the complexity of a problem is measured by its *recursion depth* (e.g., the nesting level of hierarchical structures). While length generalization tests scalability, depth generalization probes a model's capacity for compositional reasoning and handling recursive patterns, a capability central to human cognition and formal systems. Recursion underpins many fundamental domains, including **propositional logic** (Pospesel 1974) (nested quantifiers and clauses), **Boolean algebra** (Sikorski et al. 1969) (compound expressions like $(A \wedge (B \vee \neg C))$), and **recursive arithmetic** (nested operations like $3 * (2 + (5/1))$). The depth of recursion reflects the complexity of hierarchical relationships, demanding models to track intermediate states and compose operations systematically. For instance, evaluating an expression with depth k requires resolving k layers of nested dependencies, a challenge distinct from processing a flat sequence of length k.

We hypothesize that while non-recursive tasks (e.g., multi-digit addition or sequence reversal) can often be solved via transformers' autoregressive nature or enhanced positional encodings (PE), recursive problems pose a fundamentally harder challenge. Unlike linear sequences, recursive structures require *stack-like behavior*, the ability to push, pop, and backtrack through nested contexts, which transformers lack by design. Attention mechanisms, despite their global receptive field, struggle to implicitly manage dynamic stacks or resolve long-range dependencies across hierarchical layers. This limitation suggests that depth generalization may demand architectural innovations beyond standard positional biases or data scaling, such as explicit memory mechanisms or syntactic scaffolding.

Understanding this gap is essential for applications requiring rigorous symbolic reasoning, such as code generation (recursive function calls) (Allamanis et al. 2018), automated theorem proving (nested proofs) (Irving et al. 2016), or parsing (syntax trees) (Huang et al. 2018). By studying depth generalization, we aim to uncover whether transform-

ers can truly *learn recursion* from data or if they require architectural inductive biases to emulate human-like hierarchical reasoning (Wei et al. 2022).

This work aims to bridge the gap in understanding transformers' limitations on depth generalization and to develop practical solutions for this underexplored challenge. Below, we outline our contributions:

- *Diagnose the intuition behind depth generalization failure*: Investigate why transformers struggle with recursion depth (e.g., lack of stack-like mechanisms) compared to length generalization.
- *Design an effective pipeline to mitigate recursion depth decay*: Propose a method that enhances transformers' ability to handle nested structures.
- *Establish benchmarks for depth generalization*: Curate diverse datasets spanning propositional logic, Boolean algebra, and recursive arithmetic to systematically evaluate hierarchical reasoning.

2 Related Work

2.1 Length Generalization in Transformers

Transformers have shown impressive performance on many tasks, but their ability to generalize to inputs longer than those seen during training remains limited. Early work discovered that while transformers excel at short-sequence tasks like adding two numbers (e.g., "12+34"), they often fail when given longer inputs (e.g., "123456+789012") (Zhou et al. 2024). Common length generalization benchmarks (Lin et al. 2025; Xiao and Liu 2025; Cai et al. 2025; Zhou et al. 2024; Abbe et al. 2024b,a; Li et al. 2024; Anil et al. 2022) include arithmetic operations, sequence copying, and sorting - all of which test how well models can extend their reasoning to longer versions of problems they were trained on. However, these tasks typically focus on *sequential* patterns rather than *hierarchical* ones. For example, reversing a sequence requires processing elements in order, while evaluating a deeply nested mathematical expression requires understanding how operations at different levels interact.

2.2 Transformers and Recursive Structures

While length generalization has been well-studied, much less attention has been paid to how transformers handle recursive or nested structures. Recursive problems require models to track hierarchical relationships - like matching parentheses in an expression or resolving nested logical clauses. Research has shown that transformers struggle with even simple recursive patterns like balanced brackets (Dyck languages) (Bhattamishra, Patel, and Goyal 2020). Interestingly, traditional recurrent neural networks (Sherstinsky 2020) often perform better on these tasks because their architecture naturally supports the "stack-like" operations needed for recursion (Delétang et al. 2022). This suggests that the standard transformer architecture may lack crucial inductive biases needed for recursive reasoning.

Understanding how transformers process recursive structures has been an active area of research. Mechanistic studies have identified specific attention patterns that resemble stack operations, including the discovery of "deduction heads" that implement tree-climbing operations (Brinkmann et al. 2024) and induction heads that enable copying mechanisms (Olsson et al. 2022). Other work has traced how information flows through different layers during recursive tasks, revealing parallel processing motifs and depth-bounded recurrent mechanisms (Nanda, Lee, and Wattenberg 2023). These analyses reveal that while transformers can develop some strategies for handling recursion, they often do so through shortcut algorithms that fail on edge cases rather than true recursive computation (Zhang et al. 2023). Our work builds on these insights by systematically measuring depth generalization across multiple recursive tasks and proposing more robust solutions.

Recently, Looped Transformer (Yang et al. 2023; Fan et al. 2024) was proposed to improve the length generalization by adding the output at each step back to the input tokens at the vector level. It still does not address the depth generalization since it processes at the token level.

3 Three Recursive Logic Problem Instances

As most research on length generalization, we evaluate the depth generalization on *controlled* and *structured* problems. Specifically, we choose three recursive problems, each with distinct constants, functions, and evaluation goals. To match with the autoregressive nature of Transformer, all problems use postfix notation[1] (e.g., $e_1 \dots e_k f$) where expressions are evaluated step-by-step from left to right.

The Boolean Algebra evaluation problem operates on truth values where $\mathcal{C} = \{0, 1\}$ represents FALSE and TRUE, with function symbols $\mathcal{F} = \{+, *, -\}$ corresponding to OR, AND, and NOT operations respectively. For example, the postfix expression $1\,0+$ evaluates to 1. The recursive structure appears when evaluating nested expressions like $0\,1\,*\,1+$ which first computes the AND of 0 and 1, then ORs the result with 1.

For the propositional logic truth table computation, we fix two propositions p and q represented as 4-bit truth table encodings: $p = 1100$ and $q = 0101$ where bits correspond to $(T,T), (T,F), (F,T), (F,F)$ valuations. The function set $\mathcal{F} = \{+, *, -, >\}$ includes IMPLY ($>$) alongside Boolean operators. Evaluating $p\,q\,*$ yields 0100 (the AND truth table), while $p\,q\,>\,-$ computes the negation of material implication resulting in 0010. This representation preserves the recursive structure of compound formulas while operating on entire truth tables at each step.

The Compositional Arithmetic problem uses 3-digit integer constants $\{000, \dots, 999\}$ with three arithmetic operations $\mathcal{F} = \{+, -, *\}$. All operations truncate results to the least significant three digits, making $999\,001+$ evaluate to 000 due to overflow. Nested expressions like $002\,003\,*\,004+$ (equivalent to $(2\times3)+4$) demonstrate recursive evaluation. The problem focuses on recursive depth rather than digit-wise generalization of arithmetic operations.

The formal representation of these three problems are included in the supplementary material.

[1] https://simple.wikipedia.org/wiki/Postfix_notation

4 Transformers Fail Depth-Generalization

Recursive problems require sequential depth-wise computation, which poses a fundamental challenge for transformers due to their fixed-depth architecture. Unlike non-recursive tasks, such as multi-digit addition or multiplication, where positional embeddings, attention mechanisms and auto-regressive generation can effectively handle variable-length inputs, length-generalizing recursive tasks demand an unbounded depth of computation. We argue that transformers are inherently limited in their ability to generalise to recursion depths beyond the number of layers L, as each layer can only process one level of recursion.

To investigate this, we train a GPT-2 model with a binary classification head to evaluate Boolean algebra expressions. The model is trained using Boolean algebra expressions with at most 5 Boolean algebra operations from [AND, OR, NOT], until the loss stops decreasing on the in-length validation set. However, as we will demonstrate, this performance does not generalise to deeper recursion depths. This limitation arises because transformers process information in parallel across layers, while recursion requires sequential depth-wise computation, creating a fundamental mismatch.

4.1 Comparison of Depth and Length Generalization

In this work, we distinguish between length and depth when analyzing the generalization behavior of Transformers on recursive logic problems. We define **length** as the total number of Boolean algebra operators in an expression, which directly corresponds to the number of computational steps required for full evaluation while **depth** refers to the depth of the recursion tree of the expression, capturing the hierarchical structure of nested subexpressions. An illustration of this distinction is provided in Figure 1.

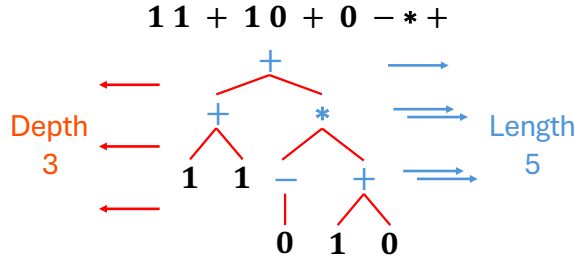

Figure 1: Definition of length and depth.

We compare the depth and length generalization capabilities of transformers in the task of Boolean algebra evaluation. The out-of-distribution test set is categorized based on the depth and length of the expressions, and the accuracy for each depth-length combination is reported in Figure 2. The results indicate that for problems with the same depth, accuracy begins to drop as the length moves just beyond the training distribution. However, as the length further increases, the accuracy does not continue to degrade monotonically but instead fluctuates, showing little correlation with length. In contrast, for any given length, the model's performance deteriorates rapidly to approximately 50% (equivalent to random guessing) as the recursion depth increases.

Depth	Length									
	5	6	7	8	9	10	11	12	13	14
4	1.00	0.86	0.70	0.66	0.67	0.80	0.67	0.74	0.70	
5	0.98	0.82	0.57	0.56	0.59	0.62	0.68	0.66	0.56	0.68
6		0.64	0.54	0.53	0.61	0.54	0.58	0.53	0.58	0.59
7			0.48	0.53	0.56	0.51	0.54	0.53	0.52	0.53

Figure 2: Heatmap of accuracies on Boolean algebra problem of different lengths and depths.

Depth	PCC (Length, Accuracy)
4	-0.3098
5	-0.1799
6	-0.1306
Average	**-0.2068**

(a) PCC between length and accuracy for depths 4 to 6.

Length	PCC (Depth, Accuracy)
6	-0.9329
7	-0.9592
8	-0.8949
9	-0.8485
10	-0.9478
11	-0.9151
12	-0.9551
13	-0.8487
14	-0.9969
Average	**-0.9210**

(b) PCC between depth and accuracy for lengths 6 to 14.

Table 1: The accuracy has a much stronger correlation with depth than with length.

We further use the Pearson correlation coefficient (PCC) (Cohen et al. 2009) to quantify the correlated relationship between accuracy and depth or length, which is defined as

$$PCC(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y},$$

where $\mathrm{Cov}(X,Y)$ is the covariance, σ_X and σ_Y are the standard deviations.

The PCC ranges from -1 to 1, where 1 and -1 indicate a perfect positive and negative relationship, respectively. $PCC = 0$ indicates no relationship.

Table 1 show that the average PCC between length and accuracy is -0.2068, while the average PCC between depth and accuracy is -0.9210, showing that accuracy has a much stronger correlation with depth than with length. This suggests that as recursion depth increases, the model's accuracy decreases more significantly and consistently, highlighting a fundamental difficulty in handling deeper recursive structures. Consequently, depth generalization poses a far greater challenge for transformers than length generalization.

4.2 Layer-Wise Analysis of Transformer Limitations

Depth	Input Tokens	Model layers					
		L1	L2	L3	L4	L5	L6
2	101*+	0.61	0.78	0.89	0.85	0.98	1.00
3	110*1*+	0.66	0.56	0.68	0.52	0.98	1.00
4	10-0*1*+	0.44	0.42	0.64	0.50	0.90	1.00
5 (Length=5)	110*-0*1*+	0.46	0.28	0.33	0.33	0.85	0.99
5 (Length=6)	11+10*-0*1*+	0.51	0.29	0.33	0.40	0.90	0.97

Figure 3: Hidden state visualizations of the last token in Boolean algebra problems with different depth, across different transformer layers.

Transformers process Boolean algebra expressions recursively, resolving intermediate operations layer by layer. The recursion problem must be solved step by step from the bottom to the top in a parse tree, where the intermediate result of each step is only accessible at the next layer. Consequently, solving a problem of depth 3 requires at least 3 layers if we assume that each layer can only resolve one step.

Since transformers have a fixed number of layers, the model struggles to reconstruct intermediate results beyond a certain depth, leading to a rapid performance drop toward 50% accuracy, equivalent to random guessing.

Also, in shallow cases, local token interactions allow the model to correctly compute sub-expressions. However, as recursion depth increases, the number of intermediate computations grows, requiring hidden states to maintain and propagate information across more layers.

We empirically study how the depth of an operator correlates with the number of layer in which it is evaluated. The first four rows in Figure 3 demonstrates how each layer's hidden state prediction changes with problem depth. The hidden state prediction of a layer is calculated by applying the output layer to the hidden state at that layer.

We observe that as depth increases, the correct prediction (in this example, 1) occurs at progressively later layers, suggesting that operations with greater depth are evaluated in later layers. In contrast, as shown in the last two rows in Figure 3, when depth remains the same but length increases by one, the hidden state prediction remains nearly unchanged. This suggests that the new operation is solved in parallel and does not require additional layers.

4.3 Limited Impact of Model Size

Our experiments reveal that model scale has surprisingly limited impact on depth generalization capabilities. As shown in Figure 4, when tested on recursion depths up to 10 (2× the maximum in-distribution depth), all model variants demonstrate nearly identical performance decay patterns. The accuracy difference between small 4-layer configurations and large 8-layer architectures never exceeds 1% at any depth level.

The consistent failure modes across scales indicate that transformers primarily rely on surface-level pattern matching rather than developing true recursive reasoning capabilities. Larger models in Figure 4 show only marginal performance improvements, suggesting they simply memorize more sophisticated patterns without fundamentally changing their approach to hierarchical structures.

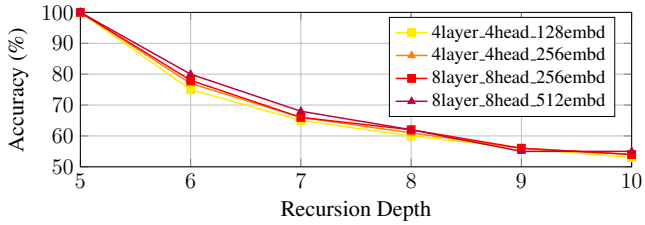

Figure 4: Different sized GPT-2 backbone models demonstrate nearly identical performance decay patterns.

The performance plateau with increased model size reveals an inherent architectural limitation. Standard transformers lack the necessary mechanisms for maintaining and resolving recursive dependencies, as evidenced by their similar error profiles on nested Boolean expressions across all model sizes. These results imply that scaling existing architectures may not solve depth generalization challenges, and that explicit architectural modifications such as memory augmentation or syntactic scaffolding may be necessary for tasks requiring genuine recursive reasoning.

5 Our Approach

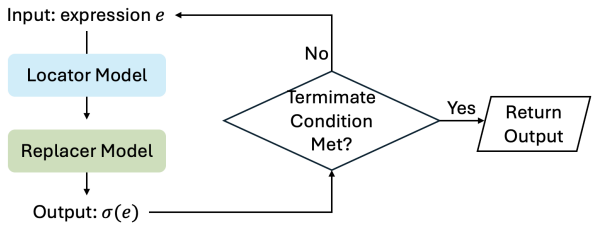

Figure 5: Overall pipeline for the looped locate-and-replace algorithm.

To address the challenge of depth generalization, we develop a novel *Looped Locate-and-Replace* (LLR) pipeline, illustrated in Figure 5 and Figure 6. The basic ides is to explicitly decomposes recursive problems into manageable subcomponents. For each input expression e, the output of passing the model in one loop is an expression $\delta(e)$, which is a reduction of e by computing the direct subexpressions.

Inside each loop, the neural network model implements two specialized components: a *locator* that identifies solvable subexpressions and a *replacer* that evaluates these components while preserving the overall structure. In our implementations, both locator and replacer are trained with a same Tranformer-based neural network: GPT-2.

While the use of the looped pipeline intuitively has a good match to recursive reasoning problems, the design of locate-replace architecture is based on an analysis that Transformer has fundamental limitation on Match and Replace problems.

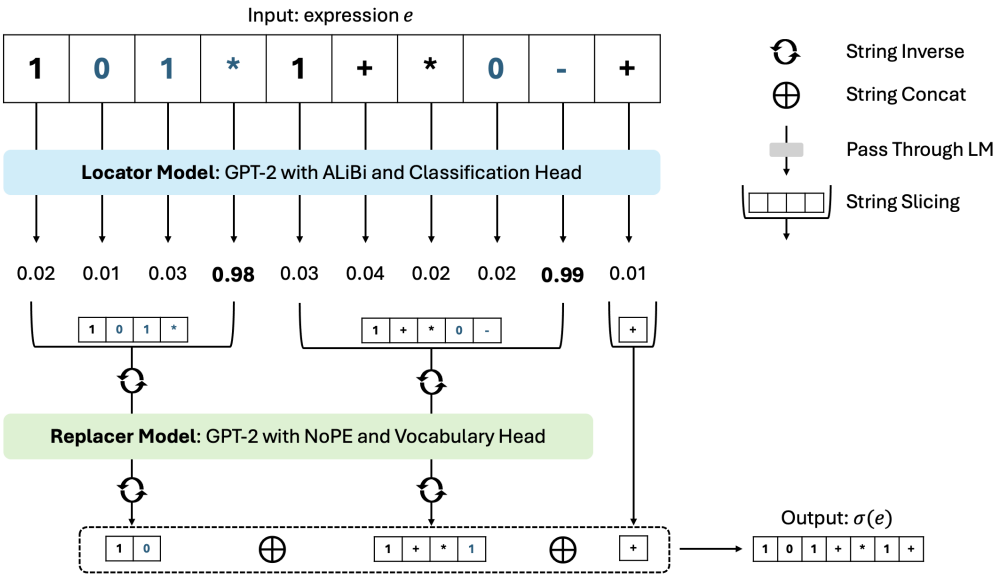

Figure 6: Overall pipeline for the locate-and-replace algorithm.

5.1 Limitations of Transformers on Match and Replace Problem

We analyze the limitations of Transformers on the *Match and Replace Problem* using the *Restricted Access Sequence Processing (RASP)* language (Weiss, Goldberg, and Yahav 2021). RASP is a computational model that abstracts the computation of Transformers into a sequence of operations on input sequences. It allows us to reason about the expressiveness of Transformers in a formal and precise way.

The *RASP-Generalization Conjecture* (Zhou et al. 2024) posits that Transformers tend to generalize well to longer input sequences on a task if the task can be solved by a short RASP program that works for all input lengths. In other words, if a task can be expressed concisely in RASP without relying on input-length-specific operations, Transformers are likely to perform well on it across different sequence lengths. Conversely, tasks that require input-length-dependent RASP programs are less likely to generalize well.

The Match and Replace Problem involves identifying a specific subsequence within a given sequence and replacing it according to predefined rules, while copying the non-pattern parts of the sequence unchanged. For example, given the input sequence `100+*`, the output should be `10*`, where the pattern `00+` is replaced with `0`, and the rest of the sequence (`1` and `*`) is copied as is. While this task appears simple, it poses significant challenges for Transformers when implemented in RASP due to the following reasons:

- **Accessing a sequence by a variable is illegal**: To match a pattern in the middle of a sequence, a loop would typically be used, requiring access to the sequence via a variable (e.g., `seq[i]`). However, this approach is illegal in RASP because Transformers cannot directly access sequence elements using variable indices. Transformers rely on attention mechanisms and positional encodings to process sequences (Kazemnejad et al. 2023), making variable-based indexing impractical.
- **Non-causal dependencies**: Without using a loop, pattern matching must be performed directly through attention mechanisms. However, matching a pattern that can appear anywhere in the sequence requires either a `shift_left` operation (to access future tokens) or attention to tokens on the right side of the current position. Both approaches are forbidden in RASP as they break the causal dependencies required by Transformers during training. A failed RASP program that illustrates this limitation is provided in the supplementary material.

According to the RASP-Generalization Conjecture, these limitations is fundamental and cannot be overcome without modifying the architecture. This analysis underscores the importance of developing new architectures that can handle non-causal tasks more effectively, potentially by relaxing the causal constraints or introducing mechanisms for lookahead. Until then, tasks like the Match and Replace Problem will remain challenging for Transformers.

We propose a two-stage locate-and-replace pipeline (Figure 6) which consists of two specialized models: (1) a locator model employing ALiBi attention with a binary classification head to detect evaluable patterns, and (2) a replacer model using no positional encoding with a standard language modeling head to generate solutions. This decoupled architecture enables precise pattern localization followed by accurate substitution.

5.2 Locator Model

To identify the boundaries of an evaluatable base-case subexpression within a sequence, we employ a locator model that outputs a scalar logit for each token in the sequence.

During the forward pass, the sequence is processed from left to right. For each token, the locator model predicts the probability that an evaluatable base-case pattern ends at that position. When the pattern boundary is reached, a distinct probability peak emerges, indicating strong confidence that the evaluable subexpression terminates at this position.

5.3 Replacer Model

After the locator model identifies evaluatable subexpressions, the replacer model computes their evaluations and integrates the results back into the sequence. As shown in Figure 6, the locator's markers enable slicing the input string into segments, each terminated by a base-case (except the final segment). The replacer processes each segment independently, generating an output sequence where: (1) base-cases are reduced to their simplified forms, and (2) non-base-case components are copied verbatim. For instance, when processing the segment `'1+*0-'`, the base-case `'0-'` evaluates to `'1'` while `'1+*'` copies unchanged, producing the intermediate output `'1+*1'`.

We employ two key optimizations to enhance this process. First, we reverse input sequences (Figure 6) to position base-cases before non-evaluatable components (e.g., `'1+*0-'` $\rightarrow$ `'-0*+1'`), which significantly improves evaluation accuracy by preventing interference from preceding non-base-case elements. The output is subsequently reversed to restore the original order. Second, following (Kazemnejad et al. 2023), we use No Positional Encoding (NoPE), which outperforms alternatives like RoPE and ALiBi for copying tasks by leveraging the transformer's pure auto-regressive nature.

The final output is constructed by concatenating all processed segments, completing a full base-case evaluation cycle σ, as demonstrated in Figure 6's bottom row.

6 Experiments and Results

6.1 Models and Datasets

The baseline model uses a GPT-2 architecture with reduced dimensions (n_embd=256, n_layer=4, n_head=4) for efficient experimentation.

Three distinct tasks described before were used for evaluation: Boolean algebra evaluation, propositional logic truth table computation, and combinatorial arithmetic. All datasets were generated recursively through automated processes, with the code available in the supplement materials. Each generated sample was classified by its recursion depth to enable depth-specific analysis. The dataset was partitioned into 9,000 training samples, 1,000 validation samples, and 1,000 test samples for each out-of-distribution depth.

6.2 Training

In formal logic tasks where precise symbolic manipulation is required, we employ **character-level tokenization** to ensure robust processing of mathematical expressions. This approach tokenizes each individual character in the input sequence separately, which prevents ambiguous token boundaries that could occur with subword tokenization, and also enables exact positional matching for pattern localization in our locate-and-replace pipeline.

For each task, we train the model on the problems with recursion at most five. All experiments were conducted with fixed random seeds to ensure reproducibility. The Adam optimizer are used with a learning rate of 5×10^{-4}. Training was performed on NVIDIA A30 GPUs with 24GB memory, using a batch size of 512 which allowed each epoch to complete in approximately 30 seconds. Early stopping was optionally employed when validation loss plateaued. The termination condition for sequence generation was task-dependent: single-character outputs for Boolean Algebra, 4-character outputs for propositional logic, and 3-character outputs for arithmetic. Given our maximum test depth of 12, generation was automatically terminated after 12 recursion iterations to prevent unnecessary computation.

For the locator model performing binary classification, we use binary cross-entropy loss with logits, defined as

$$\mathcal{L}_{\text{loc}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i))],$$

where $y_i \in \{0,1\}$ marks evaluable patterns and $\hat{y}_i$ are the logits. The replacer model uses the cross-entropy loss

$$\mathcal{L}_{\text{rep}} = -\sum_{t=1}^{T} \log p(x_t \mid x_{<t}).$$

6.3 Evaluation and Main Results

The primary evaluation metric was the accuracy of the full sequence, which required an exact match of the generated output with the ground truth. Performance was analyzed for each depth of recursion to understand how the capabilities of the model scale with the complexity of the problem.

Figure 7 demonstrates that the proposed looped locate-and-replace (*LLR*) method consistently outperforms the end-to-end vanilla transformer across all three tasks. The key observation is that the accuracy decay of LLR is significantly milder as recursion depth increases, suggesting better robustness in handling deeper recursive structures.

The end-to-end transformer exhibits a sharp accuracy drop as recursion depth increases, particularly in combinatorial arithmetic where performance quickly approaches near 0% accuracy at depth 8. In contrast, the method LLR maintains higher accuracy even at greater depths, indicating that stepwise evaluation helps mitigate error accumulation. The gradual decay of LLR may be attributed to the decreasing joint probability of correctness across all depth reduction steps. Interestingly, the decay curve does not follow a strong exponential trend, suggesting that error correlation between steps plays a role.

The Boolean algebra task outputs 0 or 1, presenting a binary classification problem. It means that when accuracy drops to approximately 50%, the model performs no better than random guessing. The method LLR maintains significantly higher accuracy (66.7% at depth 12) compared to the end-to-end approach (51.8%), reinforcing its advantage in structured reasoning tasks.

These findings suggest that LLR effectively breaks down complex recursive problems into simpler, more manageable steps, reducing error propagation.

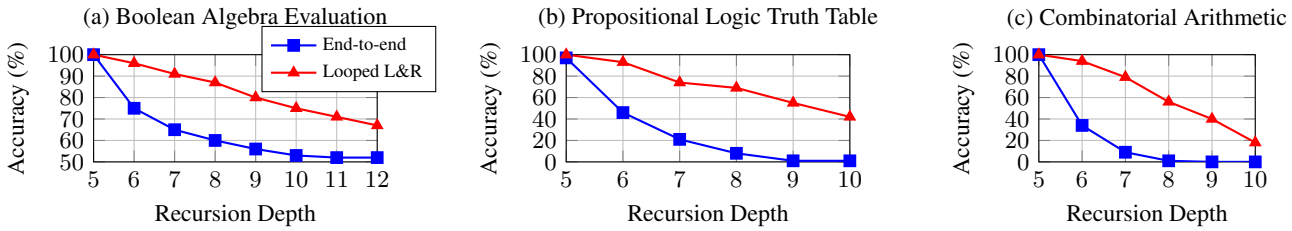

Figure 7: Accuracy decay across recursion depths. The end-to-end transformer performances are marked with blue squares and the looped locate-and-replace (*Looped L&R*) method performances are marked with red triangles.

6.4 Selection of Positional Encoding Methods

To address the challenge of base-case solving with depth generalization, we experiment with several PE methods: **absolute PE**, **rotary positional embeddings (RoPE)** (Su et al. 2021), **ALiBi (Attention with Linear Biases)** (Press, Smith, and Lewis 2021), **NoPE (no positional encoding)**, and **inverse-absolute PE**. A detailed description of each method is provided in supplementary material.

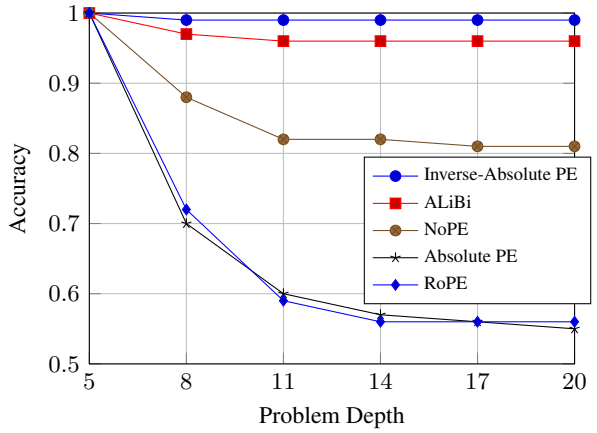

Figure 8: Evaluation of positional encoding methods

Figure 8 illustrates the experimental results. When generalized to the length of 20, Inverse-absolute PE achieves the highest accuracy at **99.9%**, followed by ALiBi at **96%**. Inverse-absolute PE performs best because the base-case pattern always appears at the rightmost position of the input sequence. By assigning a fixed positional embedding to the rightmost token, inverse-absolute PE ensures that the transformer can fit to these fixed patterns, leading to highly accurate base-case matching and replacement. We do not select this method because it does not support auto-regressive generation. In this method, the positional encoding needs to be re-computed when a new token is generated, which makes the forward pass very inefficient. In practice, we select ALiBi as the PE method for the locator model. ALiBi captures the relative positional relationships between tokens, which is beneficial for tasks requiring depth generalization. But it is slightly less robust than inverse-absolute PE because it does not explicitly enforce a fixed positional pattern for the rightmost token. It relies on relative distances, which can introduce variability in how the base-case pattern is processed.

6.5 Ablation Study

We implemented a *Looped Only* method without the *Locate* and *Replace*. Figure 9 shows that the accuracy of *Looped Only* method quickly decays to zero when the depth increases to eight. Without the *Locate* and *Replace*, the errors from each iteration will accumulate exponentially. These results justify the importance of the *Locate* and *Replace*.

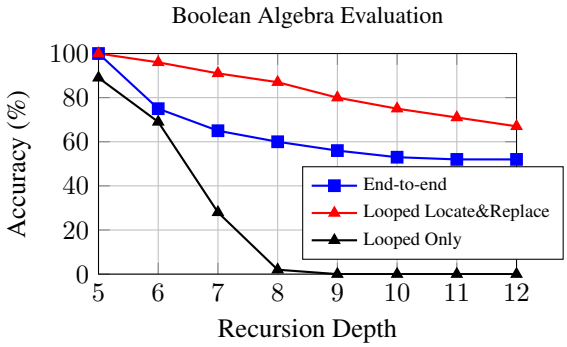

Figure 9: The accuracy of *Looped Only* (black) quickly decays with the increasing depth.

7 Conclusions

We investigated depth generalization in transformer-based large language models, a critical yet underexplored axis of generalization that evaluates their ability to handle recursive, hierarchically structured tasks. Our analysis revealed that standard attention mechanisms struggle to emulate stack-like behavior—essential for resolving deep recursion—leading to rapid performance decay as depth increases. To address this challenge, we develop a novel looped locate-and-replace pipeline that decomposes recursive problems into manageable subcomponents. We systematically evaluated depth generalization across synthetic datasets, providing empirical insights into failure modes and potential solutions. The findings underscore that the inherent architecture of transformers lacks the inductive biases required for robust recursive reasoning. By framing depth as a fundamental dimension of generalization, this work lays the groundwork for future research into more structurally aware language models.

Acknowledgments

The author gratefully acknowledges Professor Anthony Hunter for his supervision and valuable guidance during this project at University College London.

References

Abbe, E.; Bengio, S.; Lotfi, A.; and Rizk, K. 2024a. Generalization on the Unseen, Logic Reasoning and Degree Curriculum. *Journal of Machine Learning Research*, 25(331): 1–58.

Abbe, E.; Bengio, S.; Lotfi, A.; Sandon, C.; and Saremi, O. 2024b. How far can transformers reason? the globality barrier and inductive scratchpad. *Advances in Neural Information Processing Systems*, 37: 27850–27895.

Allamanis, M.; Barr, E. T.; Devanbu, P.; and Sutton, C. 2018. A Survey of Machine Learning for Big Code and Naturalness. *ACM Computing Surveys (CSUR)*, 51(4): 1–37.

Anil, C.; Wu, Y.; Andreassen, A.; Lewkowycz, A.; Misra, V.; Ramasesh, V.; Slone, A.; Gur-Ari, G.; Dyer, E.; and Neyshabur, B. 2022. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35: 38546–38556.

Bhattamishra, S.; Patel, A.; and Goyal, N. 2020. Can Transformers Handle Dyck Languages? *arXiv preprint arXiv:2006.06668*.

Brinkmann, J.; Sheshadri, A.; Levoso, V.; Swoboda, P.; and Bartelt, C. 2024. A Mechanistic Analysis of a Transformer Trained on a Symbolic Multi-Step Reasoning Task. *arXiv preprint arXiv:2402.11917*.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Cai, Z.; Lee, N.; Schwarzschild, A.; Oymak, S.; and Papailiopoulos, D. 2025. Extrapolation by Association: Length Generalization Transfer in Transformers. *arXiv preprint arXiv:2506.09251*.

Cohen, I.; Huang, Y.; Chen, J.; Benesty, J.; Benesty, J.; Chen, J.; Huang, Y.; and Cohen, I. 2009. Pearson correlation coefficient. *Noise reduction in speech processing*, 1–4.

Delétang, G.; Ruoss, A.; Grau-Moya, J.; Genewein, T.; Catt, E.; Cundy, C.; Hutter, M.; and Veness, J. 2022. RNNs Outperform Transformers on Nested Structures. *arXiv preprint arXiv:2207.13332*.

Fan, Y.; Du, Y.; Ramchandran, K.; and Lee, K. 2024. Looped Transformers for Length Generalization. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2018. Deep learning for code. *Communications of the ACM*, 61(10): 105–113.

Irving, G.; Szegedy, C.; Alemi, A. A.; Eén, N.; Chollet, F.; and Urban, J. 2016. DeepMath-Deep sequence models for premise selection. *Advances in neural information processing systems*, 29.

Kazemnejad, A.; Padhi, I.; Natesan Ramamurthy, K.; Das, P.; and Reddy, S. 2023. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36: 24892–24928.

Li, S.; You, C.; Guruganesh, G.; Ainslie, J.; Ontanon, S.; Zaheer, M.; Sanghai, S.; Yang, Y.; Kumar, S.; and Bhojanapalli, S. 2024. Functional Interpolation for Relative Positions improves Long Context Transformers. In *The Twelfth International Conference on Learning Representations*.

Lin, D.-N.; Jui-Feng, Y.; Wu, K.-D.; Xu, H.; Huang, C.-H.; and Kao, H.-Y. 2025. How Do Position Encodings Affect Length Generalization? Case Studies On In-Context Function Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 24576–24584.

Nanda, N.; Lee, A.; and Wattenberg, M. 2023. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*.

Olsson, C.; Elhage, N.; Nanda, N.; Joseph, N.; DasSarma, N.; Henighan, T.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; et al. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Pospesel, H. 1974. Introduction to logic: Propositional logic.

Press, O.; Smith, N. A.; and Lewis, M. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Sherstinsky, A. 2020. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404: 132306.

Sikorski, R.; et al. 1969. *Boolean algebras*, volume 2. Springer.

Su, J.; Lu, Y.; Pan, S.; Wen, B.; and Liu, Y. 2021. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2022. Symbolic knowledge distillation: from general language models to commonsense models. *arXiv preprint arXiv:2110.07178*.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2021. Thinking Like Transformers. In *International Conference on Machine Learning*, 11080–11090. PMLR.

Xiao, C.; and Liu, B. 2025. Generalizing Reasoning Problems to Longer Lengths. In *The Thirteenth International Conference on Learning Representations*.

Yang, L.; Zhang, Y.; Zhang, K.; et al. 2023. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*.

Zhang, S. D.; Ringer, T.; Demszky, D.; Manning, C. D.; and Potts, C. 2023. Can Transformers Learn to Solve Problems Recursively? *arXiv preprint arXiv:2305.14699*.

Zhou, H.; Bradley, A.; Littwin, E.; Razin, N.; Saremi, O.; Susskind, J. M.; Bengio, S.; and Nakkiran, P. 2024. What Algorithms can Transformers Learn? A Study in Length Generalization. In *The Twelfth International Conference on Learning Representations*.