Nested Depth Generalization in Transformers

Emile Richard
Amazon
ricemile@amazon.com
r.emile.richard@gmail.com

Abstract

Solving math problems harder than any already solved problem requires generalization of reasoning outside the training domain. This paper investigates generalization in tasks rooted in propositional logic that become increasingly challenging when expressions encapsulate more deeply nested clauses. We first evaluate general purpose language models on expressions of different depth. This shows that language models struggle with highly nested expressions, despite supervised fine-tuning, chain-of-thought reasoning with large throughput or self-reflection. We observe that small errors propagate through the nested chain of reasoning, leading to incorrect outputs. To disentangle the role of training set from implicit bias, we also train specialized models from scratch on expressions of bounded depth. Our evaluations on test sets of varying depths suggest that large models trained on large samples exhibit mild signs of depth extrapolation, but miss compact shortcut solutions that can solve the problem perfectly. For both specialized and general purpose models we demonstrate that recursive test-time inference traversing the parsed abstract syntax tree of the input unlocks depth generalization, yet remains imperfect.

1 Introduction

Assessing whether a machine can reason on problems of unbounded difficulty raises fundamental questions about experiment design and fairness of performance metrics [Bubeck et al., 2023, Trinh et al., 2024, Reid et al., 2024, Shojaee et al., 2025, Opus and Lawsen, 2025]. Measuring basic capabilities of neural networks has historically and also in the interpolation regime followed evaluations on independently and identically distributed unseen data [Vapnik and Chervonenkis, 1971, Vapnik, 1998, Bartlett et al., 2005, Belkin et al., 2019, Belkin, 2021, Montanari and Zhong, 2022]. This methodology meets its limits when assessing today's large reasoning models' performance, because (1) the internet-extracted massive training sets cover most of human knowledge, possibly resulting in leaked test sets [Mialon et al., 2023, Schaeffer et al., 2024]; and (2) the ideal test sets for reasoning tasks contain harder than training-set problems, which are by definition non-identically distributed.

To study reasoning extrapolation, training and test samples need to be split along a parameter which encodes levels of difficulty. Generalization of a model trained on easier data is measured on harder test data which is not identically distributed. This data stratification strategy is not following the i.i.d. paradigm, but has some lineage. A well studied direction of extrapolation (generalization outside the training domain) is along the context window size. Length extrapolation was hypothesized by Vaswani et al. [2017] to be well-handled by parameterized sinusoidal position embeddings. Subsequent studies of the problem (e.g. [Delétang et al., 2023]) have rejected this hypothesis, in favor of competing approaches developed using linear [Press et al., 2021], relative distance, or multiplicative position encoding Su et al. [2024]. Memory or retrieval based methods of Dai et al. [2019], Rubin and Berant [2024] and also sparse attention [Child et al., 2019] pool knowledge from chunks of text. They assume that chunks that fit in the context window are independent and distributed identically to the training data. Unfortunately, these methods fail when the input contains non-trivial connections

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

between distant tokens.
In this work we are interested in input sequences of tokens organized in hierarchies but presented to the model as a sequence. In these sequences, nested clauses create strong dependencies between far tokens. We will see that even when the test sequence fits into the context window, Transformer models fail at inducting the recursion needed to perform the target task. We demonstrate that symbolic parsing enables designing a recursion through the syntax graph that permits generalization to more deeply nested input.
A first set of experiments in Section 2 applies the proposed approach at inference time to a language model showing a $3\times$ relative performance improvement as compared to a self-reflection approach. In Section 3 we pre-train a Transformer on bounded depth (≤ 3) data and demonstrate that the resulting neuro-symbolic approach allows to achieve $\approx$90% accuracy up to depth 18 while the model's context window cannot handle expressions of depth higher than 9, where it has a near random performance of $\approx$60%.

2 Simplification of nested Boolean expressions by language models

2.1 Background on Boolean expression simplification

Simplifying a deeply nested expression has implications in logic, theorem proving [Lample et al., 2022] and circuit design as first noted by Shannon [1938]. McCluskey [1956] proposed a solution to the Boolean expression simplification problem. The Quine-McCluskey method searches the exponentially large (in the variable count) truth table for prime implicants. The search space is prohibitively large. The total number of simplified forms for an expression with k predicates is 2^{2^k}: the size of the truth table of the subsets of k entries. The recursive 'simplifying inside-out' method of Baker [1979] post-order traverses the Abstract Syntax Tree (AST) representing the expression. During the post-order traversal, simplified expressions of the children are merged using the parent node's operator. Then the resulting expression is simplified. The graph traversal algorithm visits each node of the graph once. However, because partial expressions may not simplify, the parent simplification operation may end up accumulating a higher cost than a unit cost per node. In the worst case, the combined expressions do not simplify. The resulting expressions will therefore have a length equal to the sum of the children's. This adds up to an overall exponential complexity in expression depth in the worst case scenario. While both approaches have exponential memory cost, the latter approach is more practical[1] and intuitive. In the rest of this section, we overview how LLMs approach the problem.

2.2 Description of the dataset

We use a random logical clause generator which selects uniformly from the pool of Booleans and predicates `true`, `false`, $\mathrm{p}_1, \cdots \mathrm{p}_k, \neg\mathrm{p}_1 \cdots \mathrm{p}_k, \neg\mathrm{p}_k$ (we use $k = 4$ here). Clauses are joined with operators & or | with equal probability. We generate such expressions for depths 2 to 9. Depth is defined as the maximum number of unclosed open parentheses met in the expression. We report results on 100 random samples from each depth. In Figure 3, left, we represent the character length of such expressions as a function of nested depth, which appear to grow super-linearly. For each expression, we generate a simplified version using the Python package for symbolic reasoning `sympy`. We test the validity of an expression returned by a model using 2 conditions: (1) the returned expression must be shorter than the original complex expression - note: this is a very loose requirement for simplification (2) the simplified expressions must equate the simplified expression returned by `sympy`, which we assess using `sympy.simplify(y).equals(sympy.simplify(yhat))`.

2.3 Chains-of-thoughts for recursive reasoning

We test language models using a prompt that states basic principles to use (inside-out reasoning, absorption, De Morgan law etc.) for expression simplification, see Example 1 in Appendix. We run our models on 100 samples of depths $2, \cdots, 9$ and report accuracy or rate of correctly simplified expressions. To improve performance at higher depths, we experiment with extra token budget, allowing for self-reflection, SFT, and a traversal of the parsed graph.
Maximum number of tokens. We first report results from two runs of a Claude Sonnet 3.7 model

[1] The inside-out method is implemented in `sympy.simplify`.

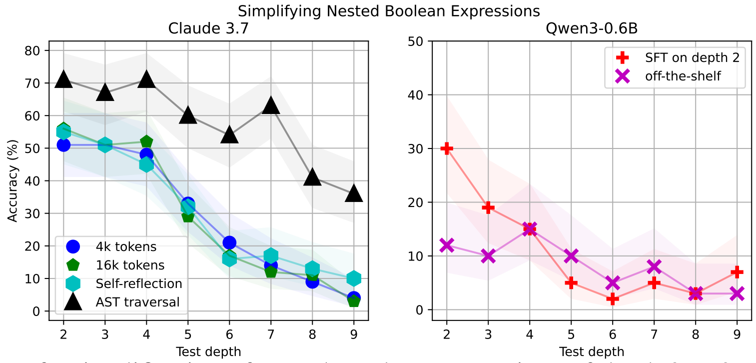

Figure 1: Left: simplification of nested Boolean expressions of depth 2 to 9 using an LLM. See Example 1 for a depth 2 case. In this experiment we vary the inference token budget / length of the output to question (reject) the hypothesis Opus and Lawsen [2025] that token budget explains collapse of model performance for complex problems. Right: Supervised Fine Tuned model on shallow (depth=2) expressions generalize to slightly deeper (depth=2, 3) but not beyond, align yet weaker than reported by Shen et al. [2023].

`"anthropic_version": "bedrock-2023-05-31"` with `max_token` variable set to 4,096 and 16,384 respectively. We observe that higher `max_token` variable does not significantly improve reasoning capability of the model as suggested by Opus and Lawsen [2025]. When inspecting failure cases (see Example 1) we see that small errors that happen throughout the chain-of-thoughts propagate and result in wrong final results. In other more complex examples such as Example 2, the model refuses to get challenged. Anecdotally, we report that Gemini 2.5 Flash Reid et al. [2024] queried on 2025-07-12 with a depth 9 example generated a 20,293 character response $\simeq 7\times$ longer than Claude's answer, yet the final response was incorrect.

Self-reflection. In a subsequent experiment, and given observations on small mistakes made by the language model, we provide the full prompt and the first answer as prompt to the same model. We also recall the desired inside-out methodology and the basic Boolean simplification rules, and ask the model to verify the reasoning and steps carefully and return the correct answer. Results of this approach barely improve on the direct inference of the model, as displayed in Figure 1 left, cyan hexagons.

AST traversal. We finally leverage the nested expression's AST[2]. to recursively call the language model while post-order traversing the expression's tree. Every time a sub-expression of depth at most 4 is met, a prompt triggers the language model to simplify the sub-expression and replace the node with it. We observe (Figure 1 left, black triangles) that this method results in a significant performance lift, for example at depth 9 where all competitor methods achieve $\leq 10\%$, the AST traversal based method is accurate in more than 40% of cases, a $4\times$ improvement. Performance boost at depths ≤ 4 is due to our implementation, which first fetches the root's children before examining the tree depth of each of the children and calling the simplifier for each.

Supervised Fine-Tuning on shallow expressions. The previous experiment tested the capability of an already trained (Large) Language Model with respect to inference budget. In a separate experiment we also examine supervised fine-tuning using a few full reasoning samples of low depth. We report results of an experiment with a `Qwen3-0.6B` model [Yang et al., 2025], fine-tuned with the chain-of-thought of the correctly simplified depth-2 samples by Claude 3.7 with `max_token`=4,096 presented in the experiment above. We observe in Figure 1 right, that while the fine-tuned model appears to be superior to the off-the-shelf model at depths 2 and 3, the performance of the two models are indistinguishably poor for depths 4 and above. This finding suggests that the Supervised Fine-Tuned Language Model was not able to generalize beyond training depth (plus one). This is to be contrasted with findings reported in Shen et al. [2023] that pre-trained models are able to *length*-generalize in arithmetic tasks.

Observations. When prompted properly, LLMs articulate a chain-of-thought which follows the inside-out or post-order traversal logic. Despite identifying a valid recursion path, we observed 3 pitfalls in the LLM's reasoning: (1) small errors often occur and propagate throughout the post-order traversal chain of reasoning, grow exponentially with the depth and super-exponentially in the number of variables, leading to incorrect conclusions, see Example 1. (2) As input length increases, the model

[2]We created the AST using the `ast` module, since `sympy`'s AST applies basic simplifications.

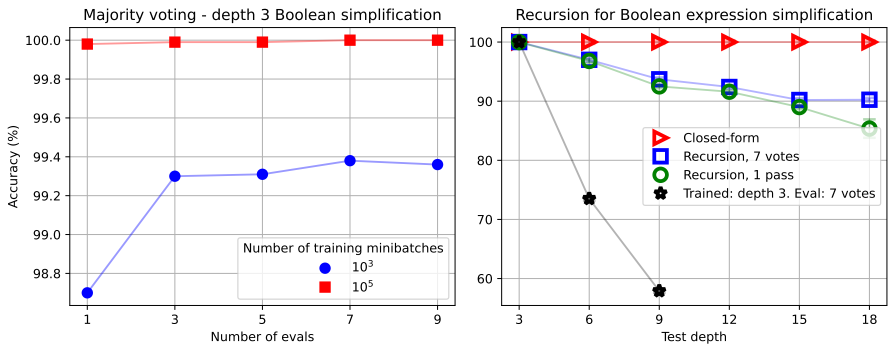

Figure 2: Boolean expression simplification. Left: Accuracy of depth 3 simplifiers trained on 10^3 and 10^5 minibatches, and with majority voting. Right: Performance in deeper test sets relying on a model trained on $q = 3$ shallow data. The model is enhanced using majority voting and recursion, and compared with a closed-form stack-augmented circuit.

appears to show increasing laziness to approach it, as suggested by the sub-linear growth of length of output against length of input reported in Figure 3. (3) Ultimately, models refuse to engage in trying to solve the problem and reject the task, as shown in Example 2.
Our experiments show that neither supervised fine-tuning nor self-reflection address the problem in a bounded number of inference calls using the internal recursions generated by the Transformer. In contrast, recursive calls that post-order traverse the AST give a performance boost by calling the neural network on shallow sub-expressions only, possibly similar to samples in the training data. How much performance boost can we get if we are certain that model calls are done on in-distribution samples only?

3 Specialized models, a closed-form oracle and test-time recursion

To disentangle Transformers' implicit bias from the train-test distribution gap, we run a set of experiments with models trained from scratch on depth-stratified train and test sets. In this section, we pre-train Transformers on a nested-depth stratified training and test set to assess whether Transformers learn parsing and recursion required for depth generalization from large training datasets. We are particularly interested in experimenting with test-time inference for models that perform well in the training domain.
Data description. We experiment on expressions formed by nested clauses of Booleans `true`, `false` joined with operators $\&$, $|$. There is no negation and no predicates in the data used here, so the output is a binary `true`, `false` and a random model has a 50% accuracy. We train a single-pass model on expressions of depth $q = 3$. For example

$$[\texttt{BOS}]\ ((\texttt{true}\&(\texttt{true}|\texttt{false}))|\texttt{true}) = \texttt{true}\ [\texttt{EOS}]$$

is one such expressions. Our test set comprises Boolean expressions of depths 3 (in sample), 6 to 18 (out-of-sample). The median length of expressions of depth 6, 9, 12, 15, 18 are respectively 87, 351, 1483, 6267, 26409 tokens.

3.1 Experiments

Pre-training and majority voting. In experiments reported in Figure 2 we use the best performing model trained on 10^5 minibatches and 7 majority votes. While we observe a performance improvement from 10^3 to 10^5 minibatches (see Figure 2 left), we do not observe grokking of a model capable of extrapolation as predicted by Murty et al. [2023a]: the best model has limited generalization, as Figure 2 right, black stars show a drastic drop in performance when depth is larger than 3. For $q \geq 12$ test sequence lengths exceed the context window length 1024 of our model. Our depth 3 trained model has 8 heads and 8 layers and embedding vectors of dimension 1024. Longer training of a model after a plateau-ed training and validation loss, and majority voting are simple mechanisms for performance optimization. We trained and compared models on 10^3 and 10^5 minibatches of 8192 Boolean expressions with 1 to 9 majority votes and present results in Figure 2 left, where we report evaluations on a 10,000 sample and a model with 7+ majority votes which seemingly performs with 100% accuracy.

Recursive post-order traversal of AST relying on a pre-trained model. We experiment with additional compute utilized at test time [Liu et al., 2025] for improving performance of a specialized model at Boolean expression simplification. Assuming that we can access a routine that extracts sub-expressions with a bounded training depth from the test sample. We implement a post-order traversal for recursive simplifications performed at the known maximum depth (here $q = 3$) of the base model. At every step, the model first identifies depth q expressions within a deeper expression. It then calls the decoder Transformer to replace the depth q expression with the simplified expression and repeats this until depth of the child to simplify decays to zero.

A closed-form model. Within the scope of the problem we study here (no negation, no predicates), we propose a closed-form neural network (details in Appendix B) that solves the problem in a single pass. The algorithm that this model's forward loop implements is an isolated case which can be written using a simple stack and as a single pass over the input sequence. It is noteworthy that the sparse (few weights) rules-based model exists. Yet it is perhaps challenging to learn such models. So while our findings seem to align at first glance with Joulin and Mikolov [2015], Grefenstette et al. [2015], Murty et al. [2023b], DuSell and Chiang [2023] that stacks can enhance the ability of neural networks to capture recursive structures; in contrast with those works (1) we did not learn stack-augmented models but wrote the closed-form expression of one as a simple proof of existence.

3.2 Observations, limitations and relation with existing work

We first observe in Figure 2, left that the direct application of the depth 3 trained model with majority voting has 100% performance in the training domain. The performance drops fast as test depth grows (at $q \leq 9$ we match 50% accuracy of a random coin flip), and fails above that (here $q \geq 12$) as the context length needed to express the input gets larger than the model's positional embedding length. The recursive program's performance decays also but more slowly with depth, from $\approx 96\%$ at $q = 9$ to $\approx 88\%$ at $q = 18$. The high quality memorizer's rare errors propagate through the inside-out chain as fast as the number of nodes in the abstract syntax tree grows (exponentially in depth) and result in amplification of errors as a function of test samples' depth. This is reminiscent of the analysis of language models' errors as in Example 1. As shows in this figure, majority voting model's performance leap increases with problem difficulty. The majority vote model performs consistently above 88% of accuracy up do depth 18. The stack-augmented closed-form model has 100% accuracy for all depths.

Not surprisingly, results above confirm that specialized Transformers can reliably handle nested expressions of the same nested depth as their training set. Zhang et al. [2024], Wang et al. [2025] reported that Transformers show basic abilities to learn recursion. Our experiments suggest that Transformers are not able to induct recursions needed to unblock depth extrapolation, nor are they grokking short-cut compact solutions. Our empirical findings suggest that increased test-time compute through a post-order traversal of the parsed AST seems to be the most promising strategy for the nested extrapolation and may hold keys to unblock extrapolation in other structured sequence problems.

4 Discussion

Our experiments with test-time inference show that recursive inference with explicit parsing and post-order traversal of the parsed Abstract Syntax Tree unlocks generalization beyond the training domain. To baseline our proposal we used a prompt which produces a seemingly correct plan or chain / tree of thoughts [Wei et al., 2022, Li et al., 2024, Yao et al., 2023, Zhou et al., 2023] for addressing complex problems (see Example 1). However, despite planning the problem breakdown correctly, the LLMs fail on deep samples. This suggests that these models do not perform recursive reasoning, perhaps because their compute budget seems to scale sub-linearly as a function of the task difficulty as demonstrated in Figure 3. When explicitly parsing the expression, recursive calls to the models ensure that (1) each sub-task receives a full token budget for inference and (2) that sub-tasks are within reach of the model's capabilities, dictated by the training domain. In this respect, depth extrapolation is fundamentally different from context length extrapolation and our method complements previous work on length extrapolation by leveraging the syntax tree of the context. Our study opens a new direction of leveraging symbolic parsing for enhancing neural models' effectiveness.

References

James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

Peter L Bartlett, Olivier Bousquet, and Shahar Mendelson. Local rademacher complexities. *The Annals of Statistics*, 2005.

Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, 2021.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the chomsky hierarchy. *International Conference on Learning Representations (ICLR)*, 2023.

Brian DuSell and David Chiang. Stack attention: Improving the ability of transformers to model hierarchical patterns. *arXiv preprint arXiv:2310.01749*, 2023.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *Advances in neural information processing systems*, 28, 2015.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

Guillaume Lample, Timothee Lacroix, Marie-Anne Lachaux, Aurelien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35:26337–26349, 2022.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 2024.

Yue Liu, Jiaying Wu, Yufei He, Hongcheng Gao, Hongyu Chen, Baolong Bi, Ruihan Gong, Jiaheng Zhang, Zhiqi Huang, and Bryan Hooi. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*, 2025.

Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6): 1417–1444, 1956.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Andrea Montanari and Yiqiao Zhong. The interpolation phase transition in neural networks: Memorization and generalization under lazy training. *The Annals of Statistics*, 50(5):2816–2847, 2022.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D Manning. Grokking of hierarchical structure in vanilla transformers. *arXiv preprint arXiv:2305.18741*, 2023a.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D Manning. Pushdown layers: Encoding recursive structure in transformer language models. *EMNLP*, 2023b.

C Opus and A Lawsen. The illusion of the illusion of thinking. *arXiv preprint ArXiv:2506.09250*, 2025.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Ohad Rubin and Jonathan Berant. Retrieval-pretrained transformer: Long-range language modeling with self-retrieval. *Transactions of the Association for Computational Linguistics*, 12:1197–1213, 2024.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2024.

Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57 (12):713–723, 1938.

Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1998.

Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Theory of Probability and Its Applications*, pages 11–30. Springer, 1971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Haozhe Wang, Qixin Xu, Che Liu, Junhong Wu, Fangzhen Lin, and Wenhu Chen. Emergent hierarchical reasoning in llms through reinforcement learning. *arXiv preprint arXiv:2509.03646*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Dylan Zhang, Curt Tigges, Zory Zhang, Stella Biderman, Maxim Raginsky, and Talia Ringer. Transformer-based models are not yet perfect at learning to emulate structural recursion. *arXiv preprint arXiv:2401.12947*, 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *ICLR*, 2023.

Appendix

Appendix A - LLM reasoning on nested Boolean expressions

Examples of prompts and answers

Example 1 Example of errors in LLM responses to a depth 2 Boolean expression simplification.

```
[Prompt] You are my math logic assistant and you are helping me solve a logic
problem. We are looking at Boolean expressions with 4 variables A, B, C, D.
Each can take values true or false. We are given a nested Boolean expression
and we would like to simplify the expression onto a shorter expression which is
equivalent to the long and complex expression. My suggestion is that you try
to use De Morgan's laws and simplify the Boolean expression starting from inside
and going out. If you write your reasoning step by step you should be able to
solve this problem. For example (A & ¬ A), i.e "A and not A" simplifies to
"false"
Here we go: can you simplifying this Boolean expression which appears between
<bool> and </bool> into its simplest irreducible form:

<bool>( ( ( ¬ C | ¬ D ) & true ) ) & ( ( ¬ C & ( B | A ) ) ) & ( ( D & ¬
C ) )</bool>
Put your final answer between <answer> and </answer> tags.
```

```
[LLM] To simplify the given Boolean expression, I will follow these steps:
1. Simplify the expressions inside the innermost parentheses using Boolean laws
and De Morgan's laws.
2. Work outwards, simplifying the expressions in the next level of parentheses
using the simplified expressions from the previous step.
3. Repeat step 2 until the entire expression is simplified.


Step 1: Simplify the innermost parentheses.
(¬C | ¬D) & true = ¬C | ¬D (since A & true = A)
¬C & (B | A) = (¬C & B) | (¬C & A)
D & ¬C = D & ¬C (no simplification)

Step 2: Simplify the next level of parentheses using the results from step
1.
((¬C | ¬D) & true) = ¬C | ¬D
((¬C & B) | (¬C & A)) = ¬C & (B | A)
(D & ¬C) = D & ¬C

Step 3: Simplify the outermost parentheses using the results from step 2.
((¬C | ¬D) & true) & ((¬C & (B | A))) & (D & ¬C)
= (¬C | ¬D) & (¬C & (B | A)) & (D & ¬C)
= (¬C & ¬D & (B | A)) & (D & ¬C)    // Error: no need to carry ¬ D
= (¬C & ¬D & B) | (¬C & ¬D & A) & (D & ¬C)
<answer>(¬C & ¬D & B) | (D & ¬C)</answer>  // Error: middle term missing

This is the simplest form of the given Boolean expression using the variables
A, B, C, and D.
// Note: the correct answer is D & ¬ C & (A | B)
```

Example 2 LLM refusal to be challenged on a depth 9 problem

```
[LLM] Okay, let's try to simplify this Boolean expression step-by-step using De
Morgan's laws and other Boolean algebra rules.
<bool>( ( ( ( ( ( ( false & ¬ A ) | ( ( ¬ C & C ) & ( ¬ C | A ) ) ) | ( ( ( (
¬ B | D ) | false ) & ( A & ¬ D ) ) & ( ( D & ¬ B ) | ( ( true & A ) | ¬ B
) ) ) ) & ( ( ( ¬ D | ¬ D ) & false ) | ( ( ¬ B | A ) | A ) ) ) | ( ( ( ( ( (
true & true ) | false ) | ( ( ( ¬ D & ¬ A ) & ( D | A ) ) & ( B | A ) ) ) &
( ( D & ¬ C ) | ( ¬ B & ¬ D ) ) ) & ( ( C | ( ¬ B | ¬ B ) ) | ( ( false | C )
| ( ( ¬ C & ¬ A ) & ( false & false ) ) ) ) ) | ( ( ( ¬ C & true ) | ( false
& true ) ) | ( ( ( ¬ C & A ) & false ) & ( ¬ C | ¬ D ) ) ) ) ) & ( ( ( ( (
D | ¬ A ) & ( ¬ B | ¬ B ) ) & ( ( ( true & true ) & ( ¬ C & ¬ D ) ) | ( (
false & D ) | ( ¬ C & true ) ) ) ) & ( ( ( ( ¬ C & C ) | ( A | ¬ D ) ) | ( (
true & ¬ C ) & ( true | ¬ C ) ) ) | ( ( ( B | B ) & ( ( A | D ) | ( C | ¬ C )
) ) & ( true & false ) ) ) ) | ( ( ( ( ( true | B ) | D ) | ( ( ( B | true ) &
( ¬ B | ¬ B ) ) | ( ¬ B | ¬ B ) ) ) & ( ( ( C & ( ¬ D & C ) ) | ( ( true | B
) & B ) ) & ( ( ( ¬ D & ¬ B ) | ( ¬ C | false ) ) & ( false & ¬ D ) ) ) ) &
( ( ( A & B ) | ( ( A | C ) | B ) ) & ( ( ( false & ¬ D ) & ( ¬ A & ¬ A ) )
| ( ( C & ( D & false ) ) & ( ¬ A | C ) ) ) ) ) ) ) )</bool>
Simplifying from the innermost parentheses:

1.  (false & ¬ A) = false
2.  (¬ C & C) = false
3.  (¬ C | A) = (¬ C | A)
4.  (false | (¬ C & C)) = false
5.  (false | (¬ C | A)) = (¬ C | A)
6.  ((¬ B | D) | false) = (¬ B | D)
7.  (A & ¬ D) = (A & ¬ D)
8.  (D & ¬ B) = (D & ¬ B)
9.  ((true & A) | ¬ B) = (A | ¬ B)
10.  ((¬ B | D) & (A & ¬ D)) = ((¬ B | D) & (A & ¬ D))
11.  ((D & ¬ B) | (A | ¬ B)) = (D & ¬ B) | (A | ¬ B)
12.  (¬ D | ¬ D) = false   // Error
13.  (false & false) = false
14.  ((¬ D | ¬ D) & false) = false
15.  (¬ B | A) = (¬ B | A)
16.  (¬ B | A) | A = (¬ B | A)

...  (continuing simplification) ...   // Refusal

<answer>(¬ C | A) | (((¬ B | D) & (A & ¬ D)) & ((D & ¬ B) | (A | ¬ B))) | (((¬
B | A) | A) | false) | (((B | A) & ((D | ¬ C) | (¬ B & ¬ D))) | ((C | (¬ B & ¬
B)) | (C | ((¬ C & ¬ A) & false)))) | ((((D | ¬ A) & (¬ B | ¬ B)) & ((¬ C &
¬ D) | (¬ C & true))) | (((¬ C | (A | ¬ D)) | ((¬ C & (true | ¬ C)) | ((B | B)
& ((A | D) | (C | ¬ C))))) & false)) | (((B | D) | ((B & (¬ B | ¬ B)) | (¬ B | ¬
B))) & (((C & (¬ D & C)) | (B & B)) & ((¬ D & ¬ B) | (¬ C | false)))) | (((A |
B) | (A | C)) & (((¬ A & ¬ A) | (C & false)) | (C & (¬ A | C))))</answer>

// Note: the correct answer is false
```

Lengths of expressions and generated answers

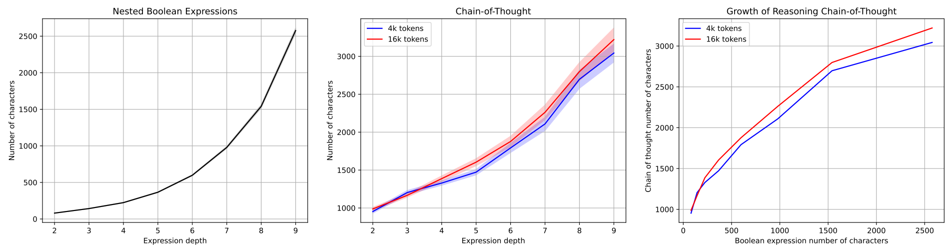

Figure 3: Number of characters in (left) complex Boolean expressions as a function of expression depth (middle) generated response as a function of depth and (right) generated response as a function of the input expression length. Note the sub-linear growth of responses lengths as a function of input length.

Appendix B - Solving Boolean evaluation in a single pass with a Neural Network

Closed-form model. We propose a closed-form stack-augmented circuit which establishes the existence of a neural-network with perfect simplification capability *in a single pass*. Our method uses a $d = 2$ dimensional state to encode counts of `true` and `&` in expressions formed by the 6 tokens: $\mathbf{E}[\texttt{true}] = (1, 0)$, $\mathbf{E}[\&] = (0, 1)$ and for $c \in \{\texttt{false}, |, (,)\}$, $\mathbf{E}[c] = (0, 0)$.

$$\mathbf{E}^{\mathsf{T}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$
$$\quad\quad\;\; \texttt{true} \;\; \texttt{false} \;\; \& \;\; | \;\; (\;\;)$$

Given a Boolean expression, the proposed method initializes a `state` variable at $(0, 0)$, the stack at empty $\texttt{stack} = []$. It then performs a single pass over the expression, see Algorithm 1. For each token in the sequence of tokens representing the Boolean expression, the state variable increments with the embedding vector $\mathbf{E}[\texttt{char}]$, then operations on the state variable or the stack are executed conditional on the token.

Why this works. The inner functioning of the proposed closed-form algorithm relies on two key observations.

1. Operator precedence: The algorithm effectively gives `&` a higher precedence than `|` without an explicit precedence table or operator stack. When an `&` is encountered, $\texttt{state}[1]$ (the `and_count`) increments. The `true_count` continues to accumulate for all subsequent literals until a closing parenthesis or a `|` is encountered. When a `|` is encountered, $\texttt{state}[0]$ (the `true_count`) is immediately capped at $\texttt{min}(1, \texttt{state}[0])$. This effectively "collapses" the OR chain so far, and any subsequent `&` or `true` will be evaluated after this collapse. This logic implicitly groups `&` operations together and treats `|` as a lower-priority separator.
2. Simplification rules. There are two simplification rules for Boolean expressions with no predicates and no $\neg$.
   (a) `true|X = true` : The $\texttt{state}[0] = \texttt{min}(1, \texttt{state}[0])$ line for `|` is a compact way of implementing the "domination" rule for OR. If any `true` is found, the `true_count` becomes 1 and stays there.
   (b) `false & X = false` : The pass for `false` and the

   $$\texttt{subexpr\_true} > \texttt{subexpr\_and}$$

   check for `&` together implement this. `false` doesn't increment the `true_count`, so if it's present in an `&` chain, the condition for the result to be `true` will fail, correctly yielding `false`.

Implementation as a neural network. The functions `phi` and `min(1, .)` in Algorithm 1 can both be implemented using feed-forward networks on a higher dimensional embedding space which also encodes the other 4 tokens for conditional operations. The function phi uses 2 layers of ReLU and 1 layer of sign (approximated with a sigmoid σ) and min(1, .), i.e. $-\texttt{ReLU}(-\mathbf{x}_0 + 1) + 1$, has a single ReLU non-linearity. We can write φ as

$$\begin{aligned}\varphi(\mathbf{x}) &= \sigma(-\mathbf{x}_1) * \psi_0 + \sigma(\mathbf{x}_1) * \texttt{ReLU}(\psi_1) \\ \psi_0 &= -\texttt{ReLU}(-\mathbf{x}_0 + 1) + 1 \\ \psi_1 &= \texttt{ReLU}(\mathbf{x}_0 - \mathbf{x}_1 + 1) - \texttt{ReLU}(\mathbf{x}_0 - \mathbf{x}_1 - 1) - 1\end{aligned}$$

The pass over the sequence sums up tokens' embeddings to update the state variable, applies feed-forward updates to the state variable and also pushes and pops elements from a stack. The model can hence be written as a uniform attention model augmented with a stack.

Algorithm 1 Single pass stack-augmented simplifier

```
def phi(true_ct, and_ct):
    if and_ct > 0: # AND: any f => f
        return int(true_ct > and_ct)
    else: # OR: any t => t
        return min(1, true_ct)

def stack_aug_simplify(expr):
    stack = [] # Stack
    state = [0,0] # true_ct, and_ct
    for char in expr: # single pass
        state += E[char]
        if char == '(':
            stack.append(state)
            state = [0, 0] # reset
        elif char == ')':
            subexpr_result = phi(*state)
            state = stack.pop()
            state[0] += subexpr_result
        elif char == '|': # any t => t
            state[0] = min(1, state[0])
    return true if state[0] > 0 else false
```