EMERGENT STACK REPRESENTATIONS IN MODELING COUNTER LANGUAGES USING TRANSFORMERS

Utkarsh Tiwari, Aviral Gupta *
Department of Computer Science
Birla Institute of Technology And Science, Pilani
{f20220052,f20220097}@pilani.bits-pilani.ac.in

Michael Hahn
Saarland Informatics Campus
Saarland University
mhahn@lst.uni-saarland.de

ABSTRACT

Transformer architectures are the backbone of most modern language models, but understanding the inner workings of these models still largely remains an open problem. One way that research in the past has tackled this problem is by isolating the learning capabilities of these architectures by training them over well-understood classes of formal languages. We extend this literature by analyzing models trained over counter languages, which can be modeled using counter v ariables. We train transformer models on 4 counter languages, and equivalently formulate these languages using stacks, whose depths can be understood as the counter values. We then probe their internal representations for stack depths at each input token to show that these models when trained as next token predictors learn stack-like representations. This brings us closer to understanding the algorithmic details of how transformers learn languages and helps in circuit discovery.

1 INTRODUCTION

Modern day language models (LMs) are increasingly capable of capturing complex sequential and linguistic patterns, achieving strong performance across diverse natural language as well as synthetic tasks. Despite their impressive capabilities and substantial amounts of effort and progress, the inner workings of these models still remains opaque and un-interpretable to a substantial extent. For instance, we know that transformer models of sufficient complexity can learn, for example, to perform modular arithmetic, but recovering the exact algorithm used by these models to perform this task remains a challenge (Nanda et al.). Building on this line of inquiry, our research focuses on formal languages, specifically examining models trained on counter languages—a class of languages formally modeled using stack memory. We demonstrate that these models develop internal representations that effectively mimic stack structures, providing insights into how they process and generate such languages.[1] Interpreting, understanding and reasoning about the algorithms and circuits within language models (LMs) remains a largely open problem, with use cases in AI safety, alignment, and performance.

Formal Languages: Formal languages provide a useful testbed to isolate and investigate the learning properties of transformers and their failure cases, since these languages have precise mathematical properties that the model can be tested on (Ackerman & Cybenko, 2020). This literature contains both empirical results (Strobl et al., 2024b; Bhattamishra et al., 2020) as well as theoretical analysis of cognitive biases and learning bounds over these formal tasks (Hahn & Rofin, 2024; Pérez et al., 2019; Zhou et al., 2023; Hahn, 2020).

*Equal Contribution.

[1] Importantly, we do not make any claims about the causality of these stacks. This is further discussed in the Section 5.

This line of inquiry is particularly significant because all algorithmic tasks can be reduced to a language within a specific class of formal languages. For instance, arithmetic in Polish notation can be modeled as a single-counter, non-regular context-free language, solvable with a stack memory but not without it. Thus, robust theoretical and empirical insights into formal languages contribute to advancing the capabilities of language models (Zhang et al., 2024). Furthermore, natural languages exhibit features that can be approximately mapped to certain classes of formal languages, often displaying recursive structures akin to those found in formal systems (Kornai, 1985; Jäger & Rogers, 2012).

Mechanistic Interpretability and Probing Classifiers: The field of Mechanistic Interpretability (MI) aims to extract the reasoning and interpretable structures present inside these models. Among other things, MI deals with identifying *features* inside language models (Rai et al., 2024). A feature is a human-interpretable property of the model's activations on specific inputs. This leads to an understanding of features as meaningful vectors in the activation space of a model. A widely used approach to understanding model structures through the aforementioned features involves *probing* these models by linking internal representations or activations with external properties of the inputs. This is done by training a classifier on these representations to predict specific properties. Known as probing classifiers (Belinkov, 2022), this framework has become a key analysis tool in numerous studies of NLP models and the methodology that we use to understand the internal structure of the trained models on these formal languages. This direction is also motivated by other works which have had success in retrieving coherent internal structures like world models in natural language (Li et al., 2021; Abdou et al., 2021) and toy/synthetic setups (Elhage et al., 2022; Li et al., 2024; Vafa et al., 2024). To our knowledge, this work is the first to leverage probing classifiers to analyze models trained on formal languages.

To more formally define the training objective of the probes, let $f : x \mapsto \hat{y}$ denote a language model, trained on a formal language, with performance measured by the language modeling task of auto-regressive next-token prediction. The model f generates intermediate representations $f_l(x)$ at layer l, called embeddings. A probing classifier $g : f_l(x) \mapsto \hat{z}$ maps these embeddings to a property z (e.g., stack depth, part-of-speech), trained and evaluated on dataset $D_P = \{(f_l(x), z^{(i)})\}$, which is formed by pairing each token's embedding with its property in that sequence. The performance of the probing classifier depends on two key factors: the probe's ability to map embeddings to the target property and the original model f's ability to generate information-rich embeddings by effectively learning the next-token prediction task. If the classifier achieves high performance, it indicates that the model has learned information relevant to the property being probed. This setup serves as a proxy for examining the internal structure of the transformer, by probing for implicit properties of the dataset it was trained on. However, the choice of property must be carefully considered to ensure meaningful and interpretable results. In our case, we adopt a more formal and structured approach by leveraging counter languages, which provide a well-defined and rigorous framework for probing. Formal languages offer a clear and systematic property to probe, grounded in their precise modeling and theoretical foundations.

2 Related Work

Transformers and Formal Language Learnability: Transformers dominate sequence modeling tasks, but their ability to model FL requiring structured memory, like counters or stacks, is only incompletely understood. While theoretically Turing-complete (Pérez et al., 2019) and universal approximators of sequence functions (Yun et al., 2020), practical learnability of FL is less understood. For example, Hahn (2020) showed Transformers struggle with Parity and Dyck-2 in the asymptotics of unbounded sequence length. Empirical studies, such as Bhattamishra et al. (2020) and Strobl et al. (2024a), demonstrate Transformers can learn Dyck-1 and Shuffle-Dyck, suggesting they can simulate counter-like behavior. These findings highlight the need for further investigation into Transformers' ability to model FL.

Probing Internal Representations: Probing classifiers have become a key tool for understanding neural model representations. By training simple classifiers on intermediate activations, researchers infer whether specific properties are encoded. For example, Voita et al. (2019) studied attention heads, while Rogers et al. (2020) and Coenen et al. (2019) explored intermediate layer information. In FL, probing has been used to analyze models trained on tasks like arithmetic and syntactic

parsing (Li et al., 2021; Abdou et al., 2021). Notably, Elhage et al. (2022) and Li et al. (2024) showed probing can reveal emergent structures in synthetic setups. Our work extends this by probing models trained on counter languages, providing insights into stack-like representations learned by Transformers.

Enforcing Stack Structures in Transformers: Several works have explored explicitly incorporating stack-like structures into neural models to handle hierarchical patterns. For example, Joulin & Mikolov (2015) proposed a neural stack for RNNs, enabling pushdown automata simulation. Similarly, Suzgun et al. (2019) introduced a differentiable stack for learning Dyck languages. In Transformers, papers like Fernandez Astudillo et al. (2020) and DuSell & Chiang (2024) explored external memory modules to enhance long-range dependency modeling. While promising, these approaches require significant architectural changes. Our work focuses on whether Transformers can implicitly learn stack-like representations without explicit constraints, offering a more interpretable approach to modeling FL.

In summary, our work builds on these foundations by analyzing Transformers' ability to model counter languages and probing their internal representations for stack-like structures. By bridging FL theory and mechanistic interpretability, we aim to advance understanding of how Transformers learn and generalize algorithmic patterns.

3 Problem Setup And Architecture

3.1 Counter Language Modeling

Counter Languages are languages modeled using counter machines, which are DFAs with counter variables which can be incremented, decremented, and set to 0. We focus our work on the Dyck language, which is a well studied class of counter languages, and the k-Shuffles of Dyck-1.

Dyck-1 is the set of well-formed parenthesis strings, is a context-free language which requires 1 counter to model it. Over the alphabet $\Sigma = \{(,)\}$ the production rules are:

$$S \to \begin{cases} \epsilon \\ (S) \\ SS \end{cases}$$

Shuffle is the binary operation $||$ on two strings which interleaves the two string in all possible ways. Inductively:

- $u \odot \epsilon = \epsilon \odot u = \{u\}$
- $\alpha u \odot \beta v = \alpha(u \odot \beta v) \cup \beta(\alpha u \odot v)$

for any $\alpha, \beta \in \Sigma$ and $u, v \in \Sigma^*$[2]. For example, the shuffle of ab, cd = $\{abcd, acbd, acdb, cabd, cadb, cdab\}$. The Shuffle operation can be extended to apply over languages $\mathcal{L}_1$ and $\mathcal{L}_2$ as:

$$\mathcal{L}_1 \odot \mathcal{L}_2 = \bigcup_{\substack{u \in \mathcal{L}_1, \\ v \in \mathcal{L}_2}} u \odot v$$

We use Shuffle-k to denote the Shuffle of k Dyck-1 languages, each with vocabulary Σ_i such that $\bigcap_{i \in [1,k]} \Sigma_i = \emptyset$, i.e. disjoint vocabularies[3].

Similar to Bhattamishra et al. (2020) Shuffle-2 is the shuffle of Dyck-1 over alphabet Σ = $\{(,)\}$ and another Dyck-1 over the alphabet Σ = $\{[,]\}$. Hence the resulting Shuffle-2 language is defined over alphabet Σ = $\{[,], (,)\}$ and contains words such as ([)] and [(()]) but not])[(.

In our experiment setup we consider 4 counter languages: Dyck-1, and Shuffle-2, Shuffle-4, and Shuffle-6. For all these languages we follow the training details from Bhattamishra et al. (2020) We

[2]The $*$ is the Kleene Star operation

[3]Hence, Shuffle-1 is the same as Dyck-1

employ an encoder-only transformer model with a linear decoder layer (language-modeling head) for sequence processing tasks. The model architecture and training details closely follow the setup from Bhattamishra et al. (2020) and is described in the Appendix.

3.2 PROBING SETUP

Shuffle-k can be modeled using k counter variables. Let s be the input string of length n, and let paren_i denote the i-th pair of parentheses with opening symbol open_i and closing symbol close_i for $1 \leq i \leq k$. During iterating over s and increasing the value of counter_i when open_i is encountered and decreasing it when close_i is encountered, if any counter value becomes negative then we can say $s \notin$ Shuffle-k. At the end of the string if the values of all counters are 0 then $s \in$ Shuffle-k.

Algorithmically, Shuffle-k can be modeled using k stacks, and pushing open_i in stack_i and popping when close_i is encountered. If any stack_i is empty when close_i is encountered then $s \notin$ Shuffle-k and if at the end all stacks are empty then $s \in$ Shuffle-k. In this formulation the depth of stack_i corresponds to counter_i. Hence, for a Shuffle-k language, we train k different probing models, one for each stack.

We probe the trained models for the depth of stacks at each input token to determine if the models learns the counter representation of these languages. We train simple feed forward networks of varying depths and complexity as multi class classifiers with ReLU activations on the internal representations from the trained model to predict the depth of stack being probed for.

The probing dataset is constructed by sampling sequences from the training corpus of the language model. It consists of 10,000 samples, each with lengths ranging between 2 and 50 tokens. The dataset is split into 8,000 samples for training and 2,000 for validation. Each sample is represented as a pair comprising the transformer encoder's embedding and the corresponding probed value. The embedding is extracted from the output of the last encoder layer of the language model and has a dimension of d_{model}. Specifically, for an input sequence of length T, the encoder produces an output of dimension $T \times d_{\text{model}}$. We slice this output along the sequence length dimension to obtain a single embedding of size $1 \times d_{\text{model}}$ for each token, which is then included in the probing dataset. This process is repeated for all samples in the dataset, ensuring that each entry captures the relevant encoder representation for probing tasks.

Keeping in the with the best practices of the literature (Hewitt & Liang, 2019) we also create a control task by randomizing the target values for the same input set. The difference between the accuracy on the main task and the control task is called the *selectivity*, and high values signify that the probing model is not learning spurious correlations, or memorizing the training data.

To further verify our probing methodology, we compile a model M using **Tracr** (Lindner et al., 2023) from the **RASP** (Weiss et al., 2021) program for Dyck-1 and use the same probing setup P on this model. Since, M is not trained using data, but is instead compiled using a known algorithm it can be used to test if P is effective at detecting stack-like features. The results of P on M are positive and are included in the Appendix.

4 RESULTS

We probe models trained on Dyck-1 for stack depth at each input token and use k probes—one per stack—for Shuffle-k models.

Figure 1 presents probing accuracies for Dyck-1 and one Shuffle-k stack, with results for other stacks in the Appendix. High probing accuracy, even with linear probes, alongside near-random control task performance suggests the probes capture genuine model representations rather than spurious patterns or memorized data.

The results indicate the presence of counter variables in models trained on counter languages. Notably, probe accuracy is higher for Shuffle-k than Dyck-1 and increases with k. This likely occurs because, as k grows, each stack updates less frequently (e.g., Dyck-1 updates every token, while each Shuffle-6 stack updates roughly every 6th token).

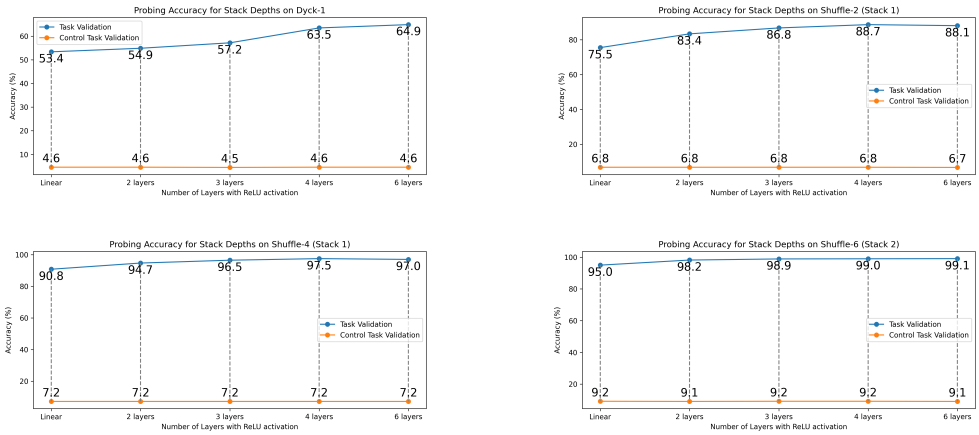


Figure 1: Probing accuracy across model architectures for different stack depths. Blue lines show task validation accuracy, while red lines represent a randomized control baseline. High task accuracy with low control accuracy indicates successful learning of stack-like structures.

5 Future Works and Conclusion

A key limitation of our study is that probing classifiers don't establish causality between detected features and model behavior. While we demonstrate the presence of counter-like structures, future work should determine if these structures fully capture the model's computational mechanisms and investigate their causal role in model outputs. The broader task of interpretability encompasses both understanding learned representations and the algorithms that operate on them - our work focuses primarily on the former, leaving algorithmic manipulation of stack structures as an important area for future research. Additional directions include exploring multiple concurrent representations and their causal relationships, studying how architectural variations impact stack formation, analyzing probing model failure cases, and validating our findings using more robust mutual information techniques.

References

Mostafa Abdou, Artur Kulmizev, Daniel Hershcovich, Stella Frank, Ellie Pavlick, and Anders Søgaard. Can language models encode perceptual structure without grounding? a case study in color. In Arianna Bisazza and Omri Abend (eds.), *Proceedings of the 25th Conference on Computational Natural Language Learning*, pp. 109–132, Online, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.conll-1.9. URL https://aclanthology.org/2021.conll-1.9/.

Joshua Ackerman and George Cybenko. A survey of neural networks and formal languages, 2020. URL https://arxiv.org/abs/2006.01338.

Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7/.

S. Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. In *Conference on Empirical Methods in Natural Language Processing*, 2020. URL https://api.semanticscholar.org/CorpusID:222225236.

Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda Viégas, and Martin Wattenberg. Visualizing and measuring the geometry of bert, 2019. URL https://arxiv.org/abs/1906.02715.

Brian DuSell and David Chiang. Stack attention: Improving the ability of transformers to model hierarchical patterns, 2024. URL https://arxiv.org/abs/2310.01749.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022. URL https://arxiv.org/abs/2209.10652.

Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. Transition-based parsing with stack-transformers. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1001–1007, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.89. URL https://aclanthology.org/2020.findings-emnlp.89/.

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

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

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks, 2019. URL https://arxiv.org/abs/1909.03368.

Gerhard Jäger and James Rogers. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367:1956 – 1970, 2012. URL https://api.semanticscholar.org/CorpusID:8765815.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets, 2015. URL https://arxiv.org/abs/1503.01007.

András Kornai. Natural languages and the Chomsky hierarchy. In Maghi King (ed.), *Second Conference of the European Chapter of the Association for Computational Linguistics*, Geneva, Switzerland, March 1985. Association for Computational Linguistics. URL https://aclanthology.org/E85-1001/.

Belinda Z. Li, Maxwell Nye, and Jacob Andreas. Implicit representations of meaning in neural language models. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1813–1827, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.143. URL https://aclanthology.org/2021.acl-long.143/.

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task, 2024. URL https://arxiv.org/abs/2210.13382.

David Lindner, Janos Kramar, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 37876–37899. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/771155abaae744e08576f1f3b4b7ac0d-Paper-Conference.pdf.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures, 2019. URL https://arxiv.org/abs/1901.03429.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. A practical review of mechanistic interpretability for transformer-based language models, 2024. URL https://arxiv.org/abs/2407.02646.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. doi: 10.1162/tacl_a_00349. URL https://aclanthology.org/2020.tacl-1.54/.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024a. ISSN 2307-387X. doi: 10.1162/tacl_a_00663. URL http://dx.doi.org/10.1162/tacl_a_00663.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024b. doi: 10.1162/tacl_a_00663. URL https://aclanthology.org/2024.tacl-1.30/.

Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M. Shieber. Memory-augmented recurrent neural networks can learn generalized dyck languages, 2019. URL https://arxiv.org/abs/1911.03329.

Keyon Vafa, Justin Y. Chen, Ashesh Rambachan, Jon Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model, 2024. URL https://arxiv.org/abs/2406.03689.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580/.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions?, 2020. URL https://arxiv.org/abs/1912.10077.

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, 2024. URL https://arxiv.org/abs/2401.12947.

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, 2023. URL https://arxiv.org/abs/2310.16028.

A COUNTER LANGUAGES

In this section we formalize the notion of counter languages by defining them as languages modeled by a k-counter machine. For $m \in \mathbb{Z}$, let $\pm m$ denote the function $\lambda x.\, x \pm m$. Let $\times 0$ denote the constant zero function $\lambda x.\, 0$. The k-counter machine is defined as [4]:

Definition 1 (General counter machine). A k-counter machine is a tuple $\langle \Sigma, Q, q_0, u, \delta, F \rangle$ with

1. A finite alphabet Σ
2. A finite set of states Q
3. An initial state q_0
4. A counter update function
$$u : \Sigma \times Q \times \{0,1\}^k \rightarrow (\{+m : m \in \mathbb{Z}\} \cup \{\times 0\})^k$$
5. A state transition function
$$\delta : \Sigma \times Q \times \{0,1\}^k \rightarrow Q$$
6. An acceptance mask
$$F \subseteq Q \times \{0,1\}^k$$

A machine processes an input string x one token at a time. For each token, we use u to update the counters and δ to update the state according to the current input token, the current state, and a finite mask of the current counter values. We formalize this in Definition 2.

For a vector $\mathbf{v}$, let $z(\mathbf{v})$ denote the broadcasted "zero-check" function, i.e.

$$z(\mathbf{v})_i = \begin{cases} 0 & \text{if } v_i = 0 \\ 1 & \text{otherwise.} \end{cases}$$

Definition 2 (Counter machine computation). Let $\langle q, c \rangle \in Q \times \mathbb{Z}^k$ be a configuration of machine M. Upon reading the input $x_t \in \Sigma$, we define the transition

$$\langle q, c \rangle \rightarrow_{x_t} \langle \delta(x_t, q, z(c)), u(x_t, q, z(c))(c) \rangle.$$

Definition 3 (Real-time acceptance). For any string $x \in \Sigma^*$ with length n, a counter machine accepts x if there exist states $q_1, \ldots, q_n$ and counter configurations $c_1, \ldots, c_n$ such that

$$\langle q_0, 0 \rangle \rightarrow_{x_1} \langle q_1, c_1 \rangle \rightarrow_{x_2} \cdots \rightarrow_{x_n} \langle q_n, c_n \rangle \in F.$$

Definition 4 (Real-time language acceptance). A counter machine accepts a language L if, for each $x \in \Sigma^*$, it accepts x iff $x \in L$.

B TRACR MODEL RESULTS

As mentioned in 3.2 we compile a model using Tracr from the following RASP program for Dyck-1[5]:

```
def num_prevs(bools) {
    prevs = select(indices, indices, <=);
    return (indices+1) *
           aggregate(prevs,
                     indicator(bools));
}
```

[4] Merrill, W. (2020). On the linguistic capacity of real-time counter automata. arXiv preprint arXiv:2004.06866.

[5] Weiss, Gail, Yoav Goldberg, and Eran Yahav. "Thinking like transformers." International Conference on Machine Learning. PMLR, 2021.

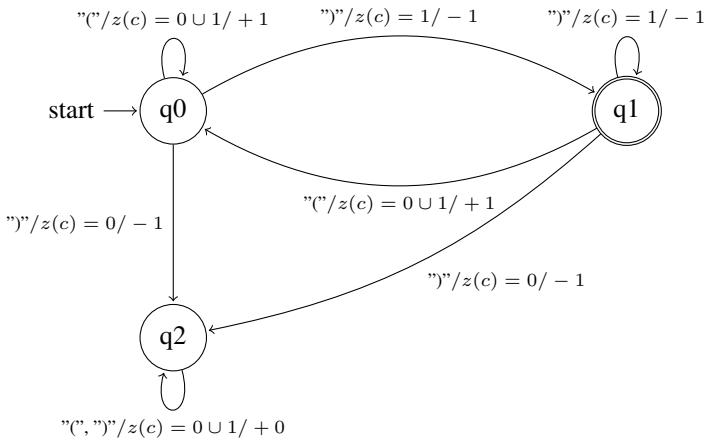


Figure 2: A graphical representation of a 1-counter machine that accepts Dyck-1 if we set F to verify that the counter is 0 and we are in q1.

$$\langle 0, q_0\rangle \underset{(}{\to} \langle 1, q_0\rangle \underset{(}{\to} \langle 2, q_0\rangle \underset{)}{\to} \langle 1, q_1\rangle \underset{)}{\to} \langle 0, q_1\rangle \in F$$

$$\langle 0, q_0\rangle \underset{(}{\to} \langle 1, q_0\rangle \underset{(}{\to} \langle 2, q_0\rangle \underset{)}{\to} \langle 1, q_1\rangle \underset{(}{\to} \langle 2, q_0\rangle \notin F$$

Figure 3: Behavior of the counter machine in 2 on $(())$ (**top**) and $(()($ (**bottom**)

```
n_opens = num_prevs(tokens=="(");
n_closes = num_prevs(tokens==")");
balance = n_opens - n_closes;
prev_imbalances = num_prevs(balance<0);
dyck1 = "F" if prev_imbalances > 0
            else
            ("T" if balance==0 else "P");
```

Here, P refers to the state when the prefix at the current token is legal but not yet balances, T when it is balanced, and F when it is illegal. This process produces a transformer model which is then used to generate a probing dataset. We leverage the same training tokens used in previous experiments to generate activation embeddings, which are then utilized to train our probing model. To robustly demonstrate the effectiveness of our probing methodology, we train the model on two distinct tasks: multi-class classification and regression. Given the transformer's unique architectural design, which is tailored to align with the underlying algorithm, our probing approach aims to recover stack-like properties inherent in the language. If successful, this would provide conclusive evidence of the efficacy of our probing setup. By successfully recovering these properties—such as stack depth—through high accuracy on the probing tasks, we demonstrate the efficacy of our probing setup. These results provide strong evidence that our approach effectively captures the hierarchical and structural features encoded in the model's representations.

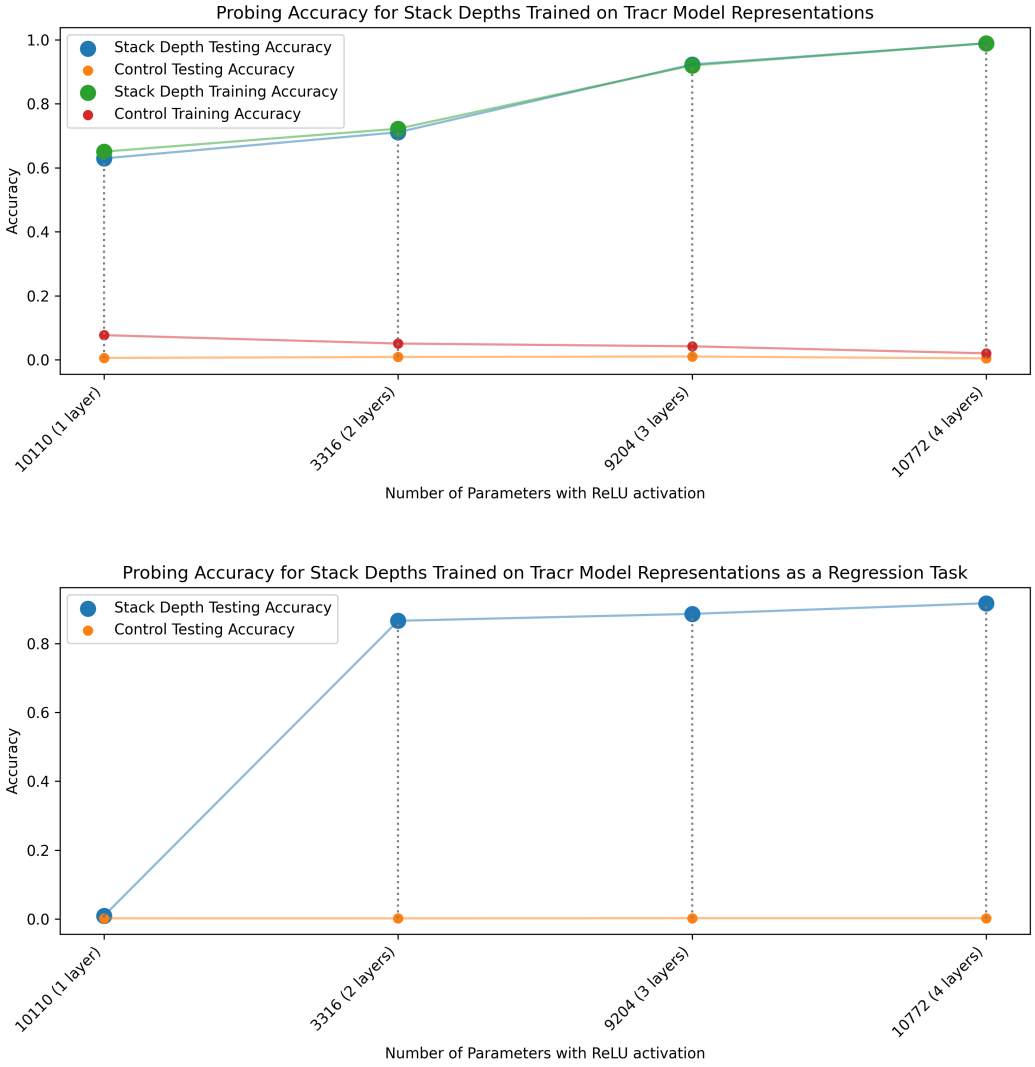


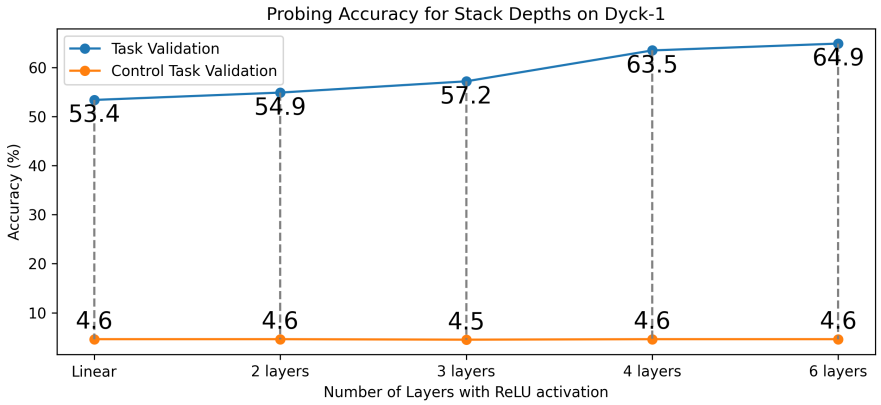
Figure 4: Probing results on the Tracr compiled model.

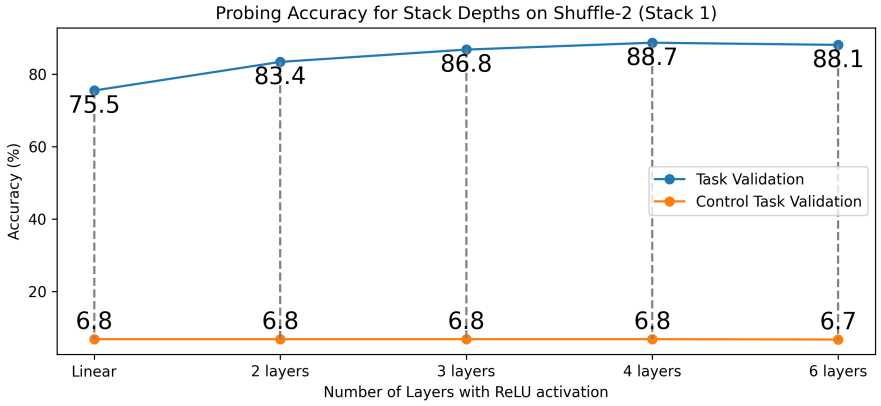
C Probing Model Results

C.1 Dyck-1

C.2 Shuffle-2

C.3 Shuffle-4

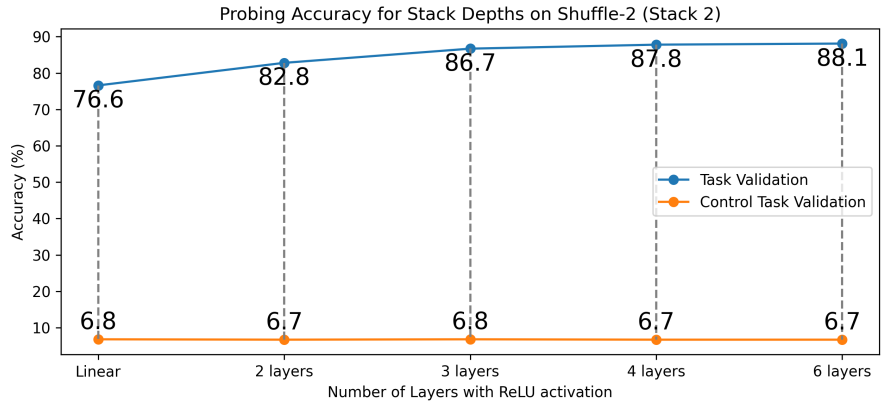


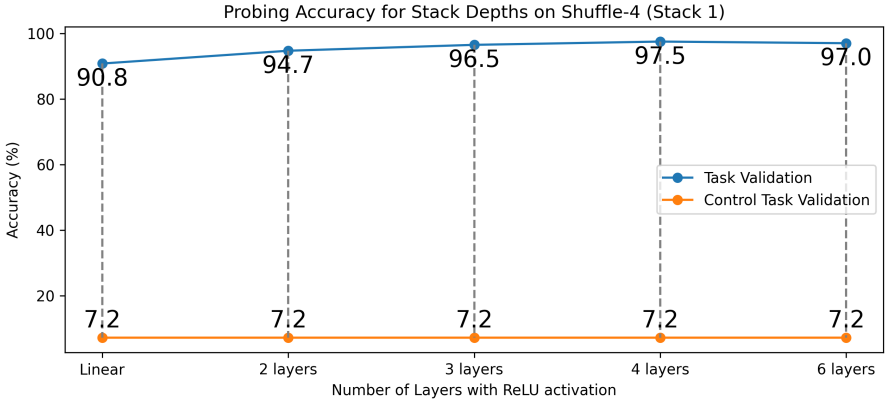


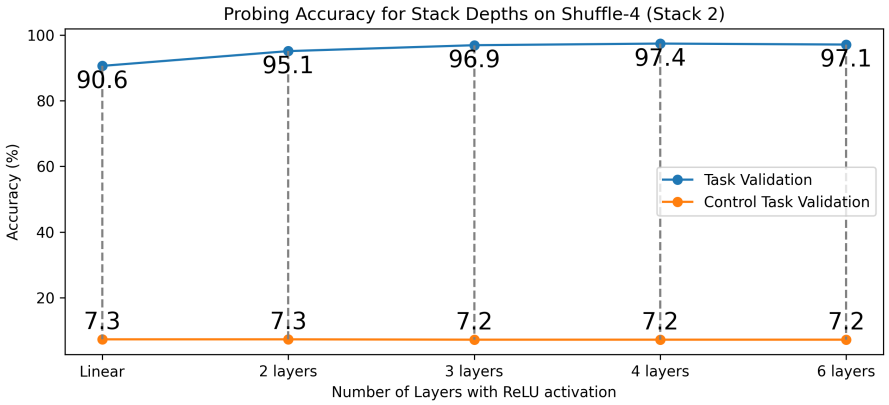
C.4 SHUFFLE-6

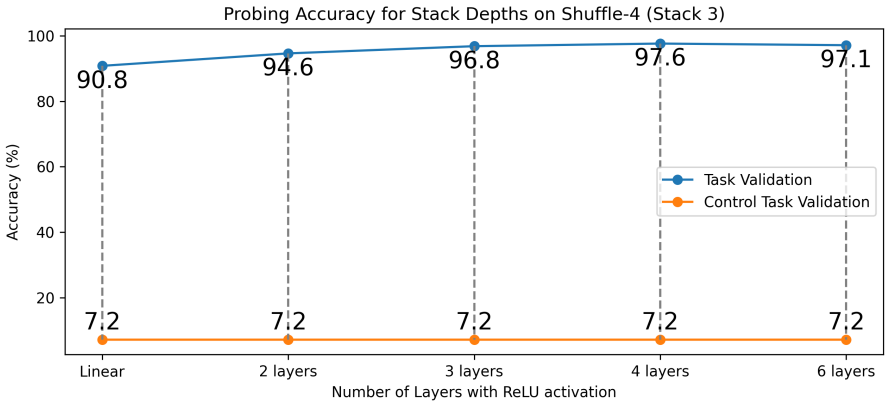
D PROBING MODEL TRAINING DETAILS

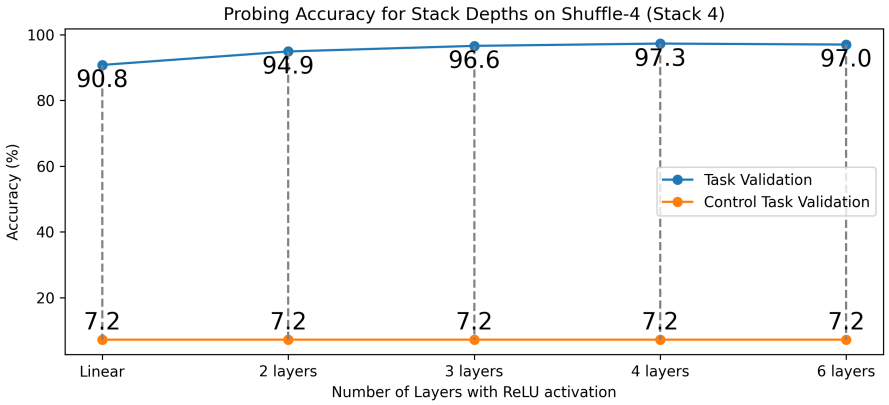
The probing classifier models are fully connected layers (ranging from linear models, to models with 6 layers with ReLU activations with hidden layer size of 128). We use dropout=0.2 and initialize the weights using Xavier initialization. The models are trained for 10 epochs with a learning rate of 0.001, batch size 32 with the Adam optimizer. We utilise the Cross-Entropy loss to train the multi-class classification task.

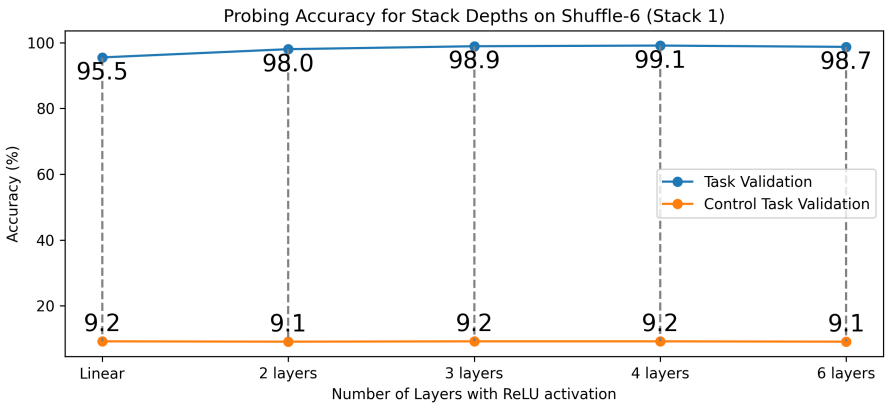

Probing Accuracy for Stack Depths on Shuffle-2 (Stack 2)
Accuracy (%)
90
80
70
60
50
40
30
20
10
76.6
82.8
86.7
87.8
88.1
6.8
6.7
6.8
6.7
6.7
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation

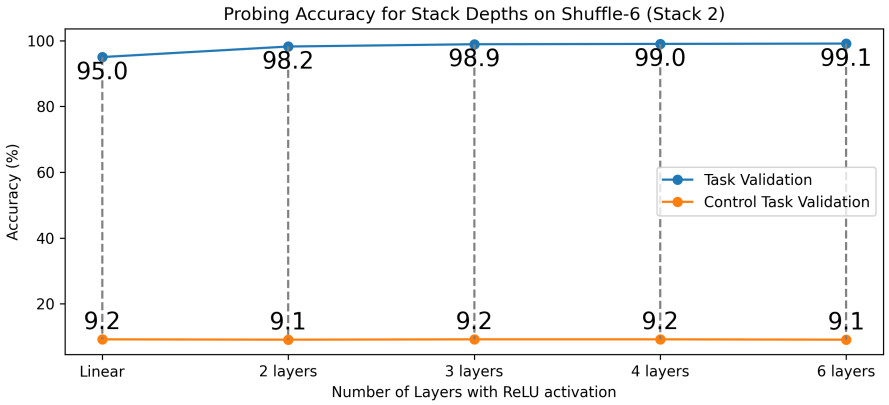

Probing Accuracy for Stack Depths on Shuffle-4 (Stack 1)
Accuracy (%)
100
80
60
40
20
90.8
94.7
96.5
97.5
97.0
7.2
7.2
7.2
7.2
7.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation

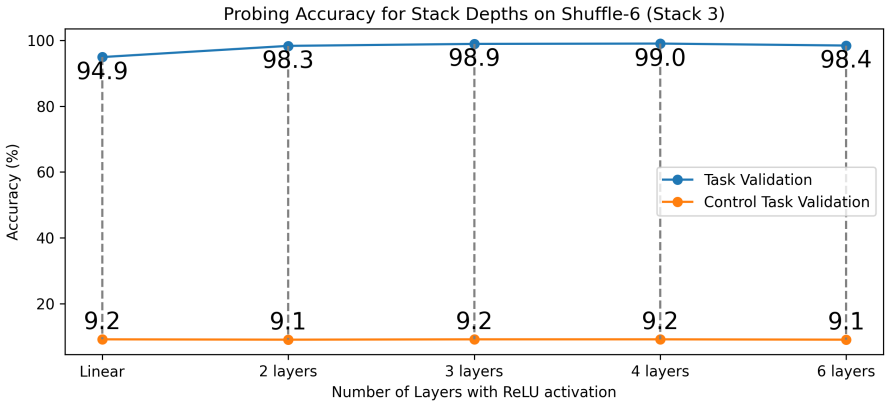

Probing Accuracy for Stack Depths on Shuffle-4 (Stack 2)
Accuracy (%)
100
80
60
40
20
90.6
95.1
96.9
97.4
97.1
7.3
7.3
7.2
7.2
7.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation

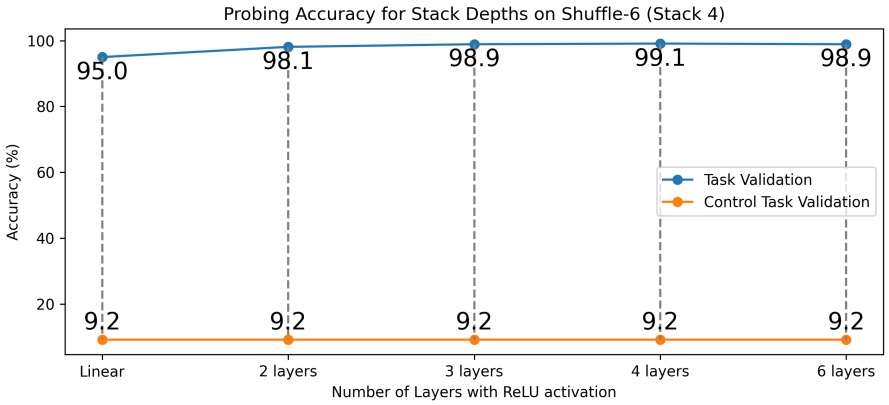

Probing Accuracy for Stack Depths on Shuffle-4 (Stack 3)
Accuracy (%)
100
80
60
40
20
90.8
94.6
96.8
97.6
97.1
7.2
7.2
7.2
7.2
7.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation

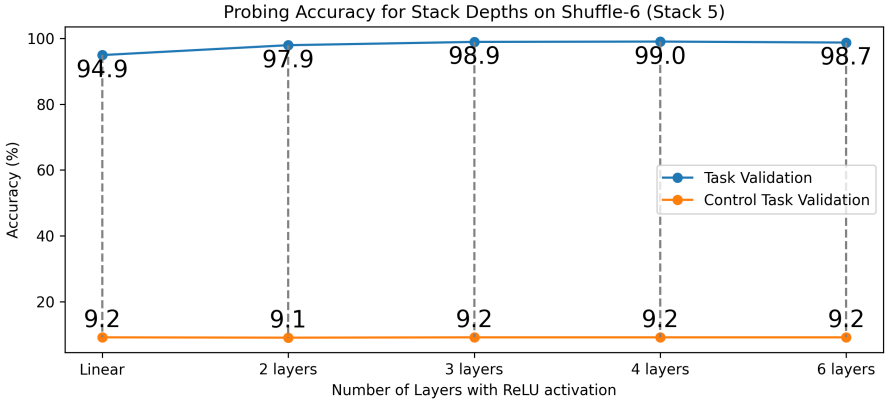

Probing Accuracy for Stack Depths on Shuffle-4 (Stack 4)
Accuracy (%)
100
80
60
40
20
90.8
94.9
96.6
97.3
97.0
7.2
7.2
7.2
7.2
7.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation

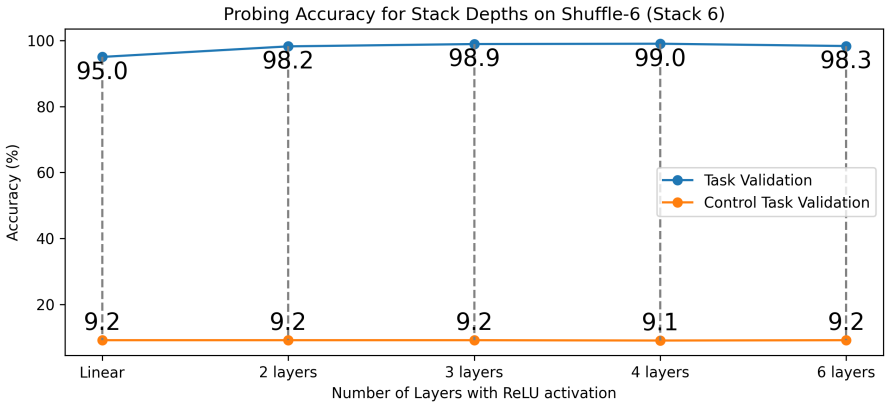

Probing Accuracy for Stack Depths on Shuffle-6 (Stack 1)
Accuracy (%)
100
80
60
40
20
95.5
98.0
98.9
99.1
98.7
9.2
9.1
9.2
9.2
9.1
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation


Probing Accuracy for Stack Depths on Shuffle-6 (Stack 2)
Accuracy (%)
100
80
60
40
20
95.0
98.2
98.9
99.0
99.1
9.2
9.1
9.2
9.2
9.1
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation


Probing Accuracy for Stack Depths on Shuffle-6 (Stack 3)
Accuracy (%)
100
80
60
40
20
94.9
98.3
98.9
99.0
98.4
9.2
9.1
9.2
9.2
9.1
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation


Probing Accuracy for Stack Depths on Shuffle-6 (Stack 4)
Accuracy (%)
100
80
60
40
20
95.0
98.1
98.9
99.1
98.9
9.2
9.2
9.2
9.2
9.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation


Probing Accuracy for Stack Depths on Shuffle-6 (Stack 5)
Accuracy (%)
100
80
60
40
20
94.9
97.9
98.9
99.0
98.7
9.2
9.1
9.2
9.2
9.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation


Probing Accuracy for Stack Depths on Shuffle-6 (Stack 6)
Accuracy (%)
100
80
60
40
20
95.0
98.2
98.9
99.0
98.3
9.2
9.2
9.2
9.1
9.2
Task Validation
Control Task Validation
Linear
2 layers
3 layers
4 layers
6 layers
Number of Layers with ReLU activation