Algorithmic Capabilities of Random Transformers

Ziqian Zhong, Jacob Andreas
Massachusetts Institute of Technology
`{ziqianz, jda}@mit.edu`

Abstract

Trained transformer models have been found to implement interpretable procedures for tasks like arithmetic and associative recall, but little is understood about how the circuits that implement these procedures originate during training. To what extent do they depend on the supervisory signal provided to models, and to what extent are they attributable to behavior already present in models at the beginning of training? To investigate these questions, we investigate what functions can be learned by randomly initialized transformers in which only the embedding layers are optimized, so that the only input–output mappings learnable from data are those already implemented (up to a choice of encoding scheme) by the randomly initialized model. We find that these random transformers can perform a wide range of meaningful algorithmic tasks, including modular arithmetic, in-weights and in-context associative recall, decimal addition, parenthesis balancing, and even some aspects of natural language text generation. Our results indicate that some algorithmic capabilities are present in transformers (and accessible via appropriately structured inputs) even before these models are trained.[1]

1 Introduction

A large body of recent work has demonstrated the effectiveness of transformer language models (LMs) [46] on general sequence-modeling tasks. Transformers seem to be especially well-suited (relative to other flexible neural models) at problems involving numerical reasoning [41, 24, 30], string manipulation [28], and various forms of in-context learning [7, 17, 1, 25]. Why is this the case?

One possibility is that some aspect of the transformer architecture makes these behaviors easy to learn. Under this hypothesis, transformer models do not implement any useful functionality when initialized; however, their loss landscape is structured such that they can be (computation- and sample-) efficiently optimized for behaviors of interest. But another possibility is that—because of intrinsic properties of the transformer architecture and parameter initialization schemes—these capabilities are *already implemented* in some fashion even in randomly initialized models.

To disentangle these possibilities, we investigate the behavior of randomly initialized transformer models in which *only the embedding layers are optimized*, leaving all other model-internal parameters fixed. If such embedding-only training is successful, it implies that the randomly initialized model's behavior on some subspace already corresponds to the input–output mapping of interest, up to a choice of encoding scheme—in other words, that the randomly initialized model can already perform the target task, and it suffices to find an encoding of inputs and outputs that induces the target behavior.

In experiments on seven tasks, we find that embedding-only training yields accurate models for a diverse set of problems spanning arithmetic, associative recall, and sequence generation—in some cases substantially outperforming similarly trained recurrent models. Remarkably, transformer *language models* trained in this fashion can even produce grammatical (though largely nonsensical)

[1]Code is available at `https://github.com/fjzzq2002/random_transformers`.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

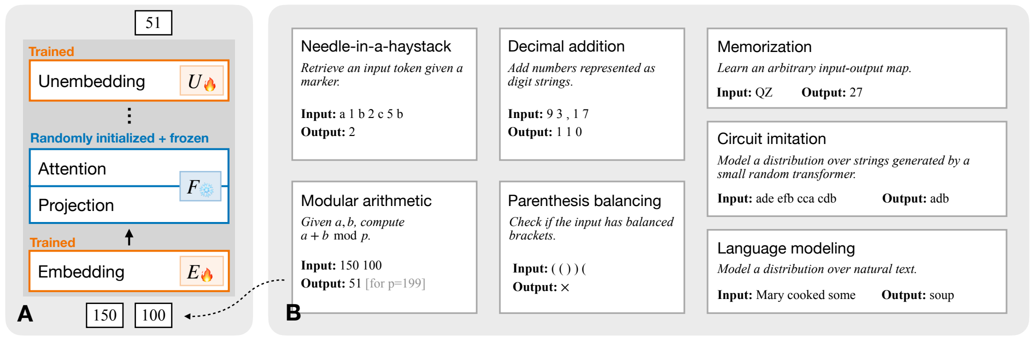


Figure 1: Overview of problem setup. **A**: Modeling approach. We initialize transformers randomly, then optimize *only* their input and output embedding layers on a dataset of interest. We find that these random transformers can be successfully trained to perform a diverse set of human-meaningful tasks. **B**: Task set. We evaluate the effectiveness of random transformers on a set of model problems involving arithmetic and memorization, as well as modeling of natural language text.

natural language text. We explain these results by showing that embedding-only training steers both inputs and model-internal representations into low-dimensional subspaces on which the model implements the target computation, a phenomenon we call "subspace selection". Embedding-only training is most successful when the target computation can be performed in a subspace that is low-dimensional relative to the ambient dimension of the model's hidden representations.

These findings build on a long line of research aimed at understanding the effectiveness of deep networks in terms of their behavior at initialization—e.g. showing that random convolutional networks are high-quality feature extractors [3, 8], or that overparameterized networks can be pruned down to sparse "lottery ticket" subnetworks that implement the correct behavior [16, 53, 39, 11]. But in contrast to past work, the solutions found by embedding-only training involve algorithmic computation rather than feature extraction, performed in low-dimensional subspaces but not by sparse sub-networks. Even more generally, our results show that pruning and optimization are not always necessary to surface useful capabilities—at least in transformers, some capabilities are available as soon as models are initialized, requiring only a learned encoding of inputs. This in turn suggests that it may be possible to partially understand the effectiveness of transformers simply by understanding their behavior at initialization.

Our work also has implications for research on circuit-level interpretability of transformers and other neural models: if even random models can perform structured, algorithmic tasks, then attempts to understand models by directly inspecting parameter matrices—and not their behavior on natural data distributions—may be fundamentally limited in their ability to characterize learned behaviors.

2 Background and Related Work

Random feature extractors Random deep convolutional networks are highly effective visual feature extractors even without training. Jarrett et al. [23] first discovered that linearly combining features from a randomly initialized one-layer convolutional network achieved comparable performance to fully trained networks for downstream vision tasks. Saxe et al. [40] showed that performance improvements from training is relatively minor comparing to architectural changes. In this work, we expanded the discussion to language models and demonstrated that training embeddings alone is sufficient to succeed in many tasks, highlighting the strong inductive bias of transformer architectures.

Neural reprogramming Neural reprogramming aims to repurpose existing neural networks for novel tasks via simple transformation layers. This technique was first purposed by Elsayed et al. [15] as a way to exploit trained neural network served by existing providers, and it was later used as a resource-efficient domain-transfer technique [45, 49]. In our work, we showed that in addition to neural networks trained for other tasks, even randomly initialized neural networks can be reprogrammed to achieve non-trivial performance.

Sparse sub-networks and lottery tickets The lottery ticket hypothesis was first proposed by Frankle and Carbin [16]: a randomly-initialized dense neural network contains subnetworks, the *winning ticket*s, that when trained in isolation can match the performance of the original network. Zhou et al. [53] and Ramanujan et al. [39] strengthened the hypothesis by discovering pruned subnetworks that achieve comparable accuracy of the trained full network within untrained, randomly initialized networks - winning tickets do not even need to be trained. The hypothesis also holds in transformers [11, 44]. Similar to the hypothesis, our work also focuses on the models' capabilities at initialization, but we showed that these capabilities can be surfaced without any pruning.

Interpreting neural circuits Many efforts have been dedicated to the interpretation of neural networks. On the arithmetic task we study in this paper, Nanda et al. [31] and Zhong et al. [52] described two different mechanisms in transformers on modular addition; Quirke and Barez [37] performed detailed mechanistic analysis for one-layer transformers trained on decimal addition. These studies provide insights into possible mechanisms transformers might employ to solve these tasks. With theoretical analysis, Wen et al. [47] proved that on the bounded Dyck task, the attention patterns and weights of neural circuits could be quite arbitrary, which we confirm in this work.

Reservoir computing Reservoir computing is a framework for computation. In the diagram, a blackbox reservoir receives data and updates its inner states there upon. A simple readout mechanism is then trained to map its inner states to the desired output. The reservior is generally kept untrained and only the readout part is trained. Under our notations, we can interpret the paradigm as training only the unembedding part of random neural networks. See Benjamin et al. [42] for a general overview for reservoir computing and Mantas et al. [29] for a survey on its applications on recurrent neural networks.

3 Setup

Models We study the behavior of decoder-only transformer language models. In these models, **inputs** x (represented as sequence of token IDs) are first assigned vector **embeddings** $h^{(0)} = E(x)$ via an **embedding layer** E. These embeddings are then passed through a series of m **intermediate layers** $F^{(1)}, F^{(2)}, \cdots, F^{(m)}$ so that $h^{(i)} = F^{(i)}(h^{(i-1)})$, with each $F^{(i)}(x)$ computing a **hidden representation** $h^{(i)}$ via a transformation:

$$h^{(i)} = F^{(i)}(h^{(i-1)}) = \text{FFN}^{(i)}(\text{SelfAtt}^{(i)}(h^{(i-1)}))\ ,$$

where FFN, SelfAtt are feed-forward and causal self-attention modules as in Radford et al. [38] (layer norms are omitted for simplicity). The final activation $h^{(m)}$ is mapped by a **unembedding layer** U to a distribution over next tokens.

To encode information about the ordering of input tokens, we implement the embedding layer E using two matrices: a token embedding matrix E_{token} and a positional embedding matrix E_{pos}. For an input $x = [x_1, x_2, \cdots, x_n]$, we first calculate the initial activations

$$h^{(0)} = E([x_1, x_2, \cdots, x_n]) = [E_{\text{token}}[x_1] + E_{\text{pos}}[1], E_{\text{token}}[x_2] + E_{\text{pos}}[2], \cdots, E_{\text{token}}[x_n] + E_{\text{pos}}[n]].$$

Similarly, the unembedding layer is parameterized by a single matrix, and model predictions have the form:

$$p(x_{n+1} \mid x_{1\cdots n}; E, F, U) \triangleq \text{softmax}\left(U h_n^{(m)}\right)[x_{n+1}],$$

where (in a slight abuse of notation) E, F and U denote embedding, intermediate, and unembedding parameters respectively.

In terms of parameters, let the hidden dimension be d, the number of layers be m, the vocabulary size be v and the maximum context length be n, the embedding layers have $\Omega((n+v)d)$ parameters as matrix E_{token} and U have shape $v \times d$ and matrix E_{pos} has shape $n \times d$, while the full network has an extra $\Omega(md^2)$ parameters.

Initialization Models are trained via gradient descent from some random initial parameterization. Following Radford et al. [38], parameters of feed-forward layers are initialized by sampling from isotropic Gaussians with mean 0 and standard deviation $0.02/\sqrt{2n}$. All the other weight matrices are initialized with 0-mean Gaussians with standard deviation 0.02. The affine transformations in layer normalizations are initialized as identity.

Training In this work, we examine language models with frozen intermediate layers, which we call **random transformers**. In these models, we fix the randomly chosen parameters intermediate layers, and train only the embedding layer E and unembedding layer U. Our experiments thus compare:

$$\textbf{Full Training:} \quad \arg\min_{E,F,U} \sum_{x,n\geq 0} -\log p(x_{n+1} \mid x_{1\cdots n}; E, F, U)$$

$$\textbf{Embedding-Only Training:} \quad \arg\min_{E,U} \sum_{x,n\geq 0} -\log p(x_{n+1} \mid x_{1\cdots n}; E, F, U)$$

where the $\arg\min$ is computed approximately via mini-batch stochastic gradient descent.

4 Random Transformers Can Perform Simple Algorithmic Tasks

Can random transformers be steered to perform meaningful tasks by optimizing only input and output tokens' embeddings? We begin by evaluating four widely-studied tasks that serve as toy models of important behaviors in large-scale LMs.

4.1 Tasks

Modular Addition This task evaluates models' ability to perform integer addition under a fixed prime modulus $p = 199$. Models receive a sequence of input tokens $[a, b]$ for $a, b \in [0, p-1]$ and must compute $(a + b) \bmod p$. When over-parameterized models are trained to perform this task, grokking (a long period of memorization followed by an abrupt transition to generalization [36]) is typically observed [26, 18]. Neural sequence models of different kinds have been found to implement two interpretable algorithms, sometimes referred to as the "Clock" [31] and "Pizza" [52], when trained to perform this task.

Needle-in-a-Haystack This task evaluates models' abilities to process long input sequences [4]. In the variant we study, models receive as input a sequence of form $[m_1, c_1, m_2, c_2, \cdots, m_k, c_k, \underline{m_u}]$. Here, $m_1, m_2, \cdots, m_k$ are distinct *markers* ($k \leq 30$) and c_i's are corresponding *values*. The input ends with a marker $\underline{m_u}$ ($u \in [1, k]$), and models must search for the previous occurrence of that marker in the input sequence and output the corresponding c_u. Specific circuits like *induction heads* are often observed in models that perform this task [34].

Decimal Addition This task evaluates models' ability to perform arithmetic operations distributed over sequences of multiple input tokens—in this case, addition of two equal-length numbers represented as digit sequences in base 10. The order of digits of both numbers and the results are reversed to simplify the task. For example, the task `39+71=110` is encoded as a pair with input `9 3 1 7` and output `0 1 1`. We use 10-digit numbers in our setup. Past work has found that fully trained models can reliably learn some versions of this task [32, 51, 43].

Parenthesis Balancing (Dyck Recognition) In this task, models are presented with a sequence of parentheses, and must predict whether they are balanced—i.e., whether the sequence contain an equal number of opening and closing parentheses, and within every prefix, the number of closing parentheses is no greater than the opening parentheses. Such sequences are also called Dyck sequences, and have been widely studied in language models because of their connection to context-free models of natural language syntax [50, 47]. Note that this task has a vocabulary of size 4 (two parentheses and two labels), so only a very small number of parameters are optimized by embedding-only training. In our setup, the input parenthesis sequences have lengths at most 60.

For the modular addition task, we partition the full set of well-formed input–output pairs into a fixed train/test split; for the other problems, we pre-generate a fixed test set but randomly generate new pairs for each training batch. Additional details may be found in Appendix D.1.

4.2 Results

Results are shown in Table 1. Here we compare random transformers with a hidden size of 1024 to fully trained models with hidden sizes of 16 and 1024. For reference, we also compare to a (fully

Task	**Random** 1024	**Random** 16	**Normal** 1024	**Normal** 16	**LSTM** 1024
Modular Addition	100.0%	1.3%	100.0%	97.2%	100.0%
Needle-in-a-Haystack	100.0%	7.5%	100.0%	12.0%	99.5%
Decimal Addition	100.0%	26.6%	100.0%	67.5%	53.0%
Parenthesis Balancing	100.0%	87.3%	92.3%	100.0%	100.0%

Table 1: Test accuracy of fully trained and random transformers, as well as fully trained LSTMs, on algorithmic tasks. Denoted numbers (1024 and 16) are hidden sizes; results are median over 10 random restarts. Random models with only trained embeddings reliably perform all four tasks, and even outperform fully trained LSTMs. See Appendix E for the accuracy curve on multiple hidden sizes.

Task	U **only**	E **only**	$\mathrm{E}_{\textbf{token}}$ **&** U **only**
Modular Addition (Train)	47.9%	68.3%	100.0%
Modular Addition (Test)	0.4%	1.2%	100.0%
Needle-in-a-Haystack	17.7%	100.0%	98.5%
Decimal Addition	27.3%	100.0%	48.5%
Parenthesis Balancing	92.3%	100.0%	100.0%

Table 2: Accuracy of embedding-only training with additional parameters fixed: optimizing only the unembedding layer, only the embedding layer, or only non-positional embeddings. Hidden sizes are all 1024; results are median over 10 random restarts. All three embedding matrices must be optimized for models to reliably complete all tasks.

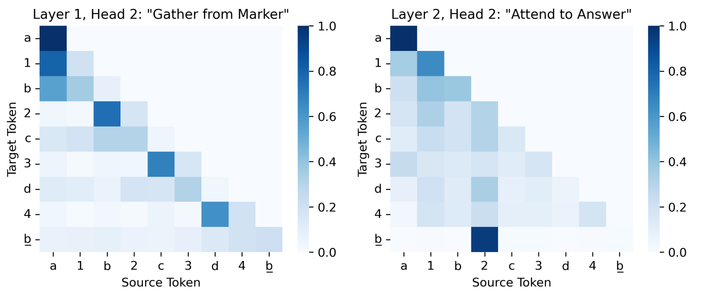
trained) LSTM, a recurrent neural sequence model [22]. All models have two hidden layers, and all results are aggregated across ten random initializations.

Random transformers learn to perform all four tasks Random transformers with trained embeddings and unembeddings obtain perfect accuracy on all four tasks consistently across restarts—sometimes outperforming *fully trained* recurrent networks (Table 1). These results thus point toward the role of a transformer-specific inductive bias in the effectiveness of embedding-only training.

In general, embedding and unembedding parameters must both be trained To further identify which model components must be optimized to obtain these results, we consider several variants of embedding-only training: (a) leaving both token and positional embeddings fixed (so only the unembedding is trained); (b) leaving the unembedding layer fixed (so only the embedding is trained); and (c) leaving positional embeddings fixed (so token embeddings and the unembedding is trained). Results are shown in Table 2. All variants fail to reach perfect accuracy on at least one task. Notably, the variant that only trains the unembedding is unable to reach near-perfect accuracy on *any* task.

Random transformers exhibit interpretable attention patterns A closer examination of the trained random models reveals similar mechanistic behaviors to their fully trained counterparts. For example, we observe attention patterns similar to "induction heads" [34] previously described in fully trained models for associative recall tasks (Figure 2).

Random transformers use structured embeddings In the modular addition task, learned embeddings form circles in low-dimensional subspaces, another phenomenon observed in fully trained models for these tasks [26, 31] (Fig. 3). To better understand similarities between these models and their fully trained counterparts, we also computed the *distance irrelevance* and *gradient symmetricity* metrics described by Zhong et al. [52] for distinguishing between networks that perform modular arithmetic via the "Clock" or "Pizza" algorithm. We find a gradient symmetricity of 0.88 and a distance irrelevance of 0.88, consistent with a Clock-like solution.



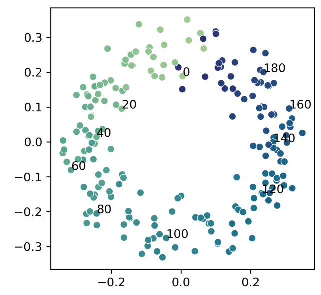


Figure 2: Attention patterns observed in a 2-layer 1024-width random transformer trained on the needle-in-a-haystack task. The input sequence is `a 1 b 2 c 3 d 4 b̲`. The layer-1 head is used by values to attend to their markers, and the layer-2 head is used by the query to attend to its associated value.

Figure 3: Circular embedding observed in random transformer on modular addition. We plot the projection of the embedding matrix onto its first and third principal components, with tokens colored according to their numeric value.

Transformer	**Accuracy**	**Memorized bits**	**Trainable parameters**	**Bits per parameter**
Normal	80.54%	1900177	659584	2.88
Random	4.53%	106876	262784	0.41

Table 3: Comparison of normal and random transformer in the memorization task.

5 Random Transformers Can Memorize and Generate Structured Sequences

The preceding experiments evaluated the ability of random transformers to implement single, highly structured input–output mappings. Can these models scale to more challenging tasks, involving memorization of arbitrary associations or even free-form text generation?

5.1 Memorization

Past work by Allen-Zhu and Li [2] has found that fully trained transformers can store roughly two bits of input per parameter. We investigate whether a similar scaling trend holds for random transformers. We study a simple memorization task in which we generate a random mapping from a two-integer key to a one-integer value, with all integers ranging from 1 to 512. Such a function requires 9 bits per input–output mapping to represent ($\log_2 512 = 9$), and may be defined for up to 262144 ($= 512^2$) values. Unlike the algorithmic tasks above, here the learned function *must* be fully specified by embeddings rather than the pre-trained model, and these experiments mainly evaluate how efficiently information can be stored in these embeddings.

We evaluate fully trained and random transformers of width 128 and 2 layers (Table 3). We measure success using an exact-match metric—an input–output pair is considered to be successfully memorized if the output token assigned highest probability by the model matches the training data. Fully trained transformers memorized 80% of training examples, stored 2.9 bits per parameter, while random transformers memorized only 5% of examples, corresponding to 0.4 bits per *trainable* parameter.

5.2 Language Modeling

Modeling natural language requires both memorization of arbitrary associations (e.g. between words and their parts of speech, as in Section 5.1), and structured sequence generation procedures (e.g. to enforce subject–verb agreement and close quotation marks, as in Section 4). Can random transformers make any headway on this task?

We train models on the `TinyStories` dataset, a collection of easy-to-understand stories generated by GPT-3.5 and GPT-4 [14] which have been shown to induce fluent text generation when used to train

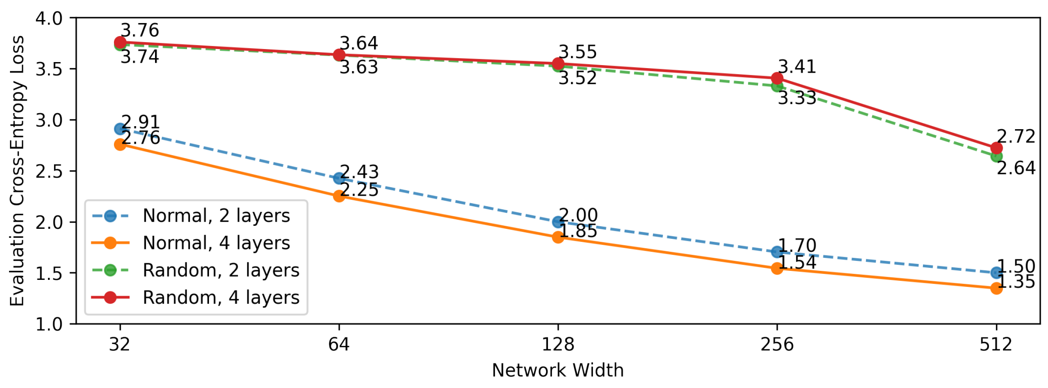


Figure 4: Language modeling performances (measured in cross-entropy loss or equivalently log perplexity, lower is better) for fully trained and random transformers. Comparatively large hidden sizes are needed for random models to match the performance of fully trained models.

Once upon a time, there was a little boy named Max. Max loved to play with his toy blocks. He had blocks of all colors and shapes. Max liked to build tall towers and big castles. One day, Max was playing with his blocks in his room. He built the tallest tower he had ever made. It was so tall that it almost touched the ceiling! Max was very proud of his tower. Suddenly,

Reference (GPT-4o): ... Suddenly, Max's cat, Whiskers, ran into the room. Whiskers saw the tall tower and thought it was a fun thing to play with. He jumped up and knocked the tower over. The blocks fell all over the floor. Max was very sad. He had worked so hard to build the tower, and now it was gone.

Fully Trained Transformer: ... Suddenly, Max's little sister came into the room. She wanted to play with Max's blocks too. But Max didn't want to share. He wanted to keep playing with his blocks. His sister started to cry and Max didn't understand why his sister was so upset. He thought he was just playing with them. So, he went to his room and looked for the blocks. After a while, he found them under the bed. Max was so happy that he hugged his sister and his sister. They played together and had a lot of fun. The end.

Random Transformer (Completion 1): ... Suddenly, Max started to blow. Max saw a loud noise coming from the tower. Max's tower was very high and strong. Max wanted to play with his blocks. Max was very proud of himself for being so helpful. Max decided to take care of his tower and his tower. Max was so happy that he had done. Max wanted to keep himself. He had to be so much fun and proud of himself. Max was happy to be able to help. Max was able to save the tower and help his tower. Max was so proud of himself. Max was happy to have such a good tower and could do it.

Random Transformer (Completion 2): ... Suddenly, Max's tower fell down and landed on his floor. Max fell down and hurt his knee. Max cried and came to him. Max was very sad and didn't want to get hurt. Max's feelings. Max didn't want to get away. Max just wanted to make it so badly! Max tried to take it to break. Max put him back on the floor. Max tried to make up but Max didn't know what to do. Max was wrong. Max said, "Don't worry, Max. Max are very bad and Max. Max didn't want to share it with Max. Max to play with Max and Max. Max was very sad. Max loved to play with his friends and Max played together and had lots of fun. Max played together all day. Max were happy that Max learned that sharing is caring. Max learned that being selfish and always had learned that sometimes things can share and others can make others happy again.

Figure 5: Sample completion generated by fully trained and random 512-width 2-layer transformers. While random models produce less coherent than fully trained models, they nonetheless generate text that is largely grammatical topically appropriate.

smaller models. As in previous sections, we evaluate both embedding-only and full training with 2- and 4-layer transformers with various hidden sizes. Scaling curves are shown in Fig. 4. Fully trained models obtain a cross-entropy loss of 1.35, on par with the results reported by Eldan and Li [14]. Our trained random transformer with 512 width and ~10M trainable parameters achieved a cross-entropy loss of 2.64, roughly on par with the fully trained ones with 32 width and only ~0.7M trainable parameters. Moreover, adding additional hidden layers does not appear to improve performance performance—2-layer random transformers in fact achieve better losses than 4-layer models.

Example outputs sampled from these models (on a newly generated prompt) are shown in Fig. 5. Even though random models obtain significantly worse perplexity than fully trained models, they could still perform several key sub-tasks needed for language generation: generated sentences are generally grammatical, topically coherent with the prompt, correctly resolve some long-distance dependencies (e.g. references to the name *Max*) and perhaps even high-level narrative structure.

6 Random Transformers Operate in Low-Dimensional Subspaces

Can we explain the success of random transformers in the tasks studied above? In this section, we present evidence that embedding-only training steers the hidden computation in transformers into low-dimensional subspaces in which target functions are already implemented. We term this phenomenon **subspace selection**, and show that it is distinct from *sparsification*, as these subspaces are distributed across neurons.

In Section 6.1, we measured fraction of activation variance explained by top principal components in various tasks. For algorithmic tasks, we show that both normal and random transformers work in low-dimensional subspaces, which are sufficient for solving these tasks (Appendix F). However, for language modeling and memorization, the random transformers displayed more subspace selection compared to the fully trained ones, and as a result, they attained lower performances. In Section 6.2 we constructed a task that explicitly requires operating on high-dimensional spaces, circuit imitation, and indeed, the random transformers exhibit significant performance gap compared to normal transformers.

6.1 Low-Dimensional Hidden Representations in Algorithmic and Memorization Tasks

To characterize the geometry of random transformers' internal representations, we present models (trained for all previously described tasks) with a set of randomly chosen inputs and collect their embeddings and hidden representations of these inputs at different layers. Using these representations, we perform two analyses: (1) the fraction of variance explained by the top **principal components** of these hidden representations (which will be large if representations lie in a low-dimensional subspace), and (2) the fraction of variance explained by the most variable entries in hidden state vectors, or **neurons** (which will be large if computation is confined to a sparse sub-network).

Both fully trained and random transformers exhibit subspace selection but not sparsification (Table 4 top) in the four algorithmic tasks. In Appendix F, we show that this behavior is expected, insofar as all four algorithmic tasks can be solved by shallow transformer-like circuits that operate in low-dimensional subspaces. On memorization and language modeling tasks, random transformers become much more concentrated on a small subspace than fully trained transformers, thus using a lower effective dimension (Table 4 bottom and Table 5). In the language modeling task, more than 30% of variance in hidden representations is explained by 10 components.

6.2 Subspace Selection in Circuit Imitation

To provide another window into these results, we characterize *how large* the hidden representations of a random model must be for it to simulate a random circuit that operates in a low-dimensional subspace.

To do so, we first generate a small, random *target* transformer associated with a distribution over strings $\tilde{p}$, then perform embedding-only training in a different, randomly initialized transformer to simulate its behavior on some domain of interest by minimizing:

$$\arg\min_{E,U} \ \mathbb{E}_x[\mathrm{KL}(\tilde{p}(\cdot \mid x) \parallel p(\cdot \mid x; E, F, U)]$$

To construct target models, we begin with the same initialization scheme described in Section 3, then we scale the query and key parameters in the attention mechanism by a factor of 10, and and the feed forward weights and biases by a factor of 20. We also scale the final projection layer by $100/\sqrt{\text{width}}$. (This initialization scheme increases variability of attention patterns and target model predictions across random restarts; see Appendix D.2.2 for additional discussion.)

Task and Transformer Type		Principal Component Basis			Neuron Basis		
Task	**Model**	**Emb**	**L1**	**L2**	**Emb**	**L1**	**L2**
Modular Addition	Normal	42.0%	21.5%	13.8%	3.8%	1.6%	3.4%
	Random	72.9%	63.6%	44.7%	2.8%	3.3%	2.9%
Multi-digit Addition	Normal	45.3%	87.0%	87.3%	1.7%	3.3%	2.6%
	Random	55.1%	72.7%	63.5%	2.0%	3.9%	2.8%
Needle-in-a-Haystack	Normal	35.5%	83.5%	47.7%	1.5%	5.2%	2.8%
	Random	31.3%	30.0%	21.5%	1.8%	1.8%	1.6%
Balanced Parentheses	Normal	72.0%	99.4%	98.2%	2.9%	7.8%	18.0%
	Random	74.9%	85.8%	80.9%	4.9%	4.0%	3.7%
Memorization	Normal	15.0%	25.3%	23.4%	10.2%	12.2%	10.5%
	Random	27.5%	27.5%	24.7%	9.9%	9.8%	9.7%

Table 4: Median explained variance from top 10 directions under principal and neuron basis. Activations after embedding (*Emb*), layer 1 (*L1*) and layer 2 (*L2*) are collected from multiple trained 2-layer models of width 1024 and 128 (for memorization). *Normal* transformers are fully trained while *Random* transformers have only embedding and unembedding layers trained, as in previous experiments. Across tasks, a large fraction variance in models' hidden representations is explained by a small number of principal components, but these components do not appear to be aligned to individual neurons or sparse sub-networks.

Task and Transformer Type		Layer				
Task	**Model**	**Emb**	**L1**	**L2**	**L3**	**L4**
Language Modeling	**Principal Component Basis**					
	Normal	29.5%	27.3%	24.3%	23.2%	17.6%
	Random	43.0%	30.2%	31.8%	32.1%	31.7%
	Neuron Basis					
	Normal	8.6%	16.4%	14.7%	13.9%	5.9%
	Random	3.3%	2.9%	2.8%	2.9%	2.9%

Table 5: Median explained variance from top 10 directions under principal and neuron basis for language modeling task. Activations after embedding (*Emb*) and after every layer (*L1, L2, L3, L4*) are collected from trained 4-layer models of width 512. As above, a substantial fraction of variance is explained by a small number of principal components, especially in random transformers.

We evaluate fully trained and random transformers' ability to fit the target distribution for target models with a 512-token vocabulary, three layers, two attention heads, and varying hidden dimensions. Training inputs x consist of 40-token sequences generated uniformly at random.

Results are shown in Fig. 6. In general, random transformers can only match the behavior of shallower (3 vs 1) and or significantly narrower (512 vs 128) models, with a sharp increase in error moving from 12 $\rightarrow$ 16 $\rightarrow$ 32-dimensional models, suggesting that random transformers may *only* be able to learn computations that can be performed in lower-dimensional subspaces.

7 Discussion

We have shown that transformer sequence models can accomplish a variety of meaningful tasks when only their embedding layers are optimized. For tasks involving memorization, these results show that much of models' "knowledge" can be encapsulated by input embeddings rather than models' internal parameters. For more algorithmic tasks like arithmetic and parenthesis balancing, which require relatively sophisticated circuits to perform, our experiments show that versions of these circuits can be accessed in random transformers (but not LSTM sequence models) simply by constructing appropriate input and output embeddings that confine models' internal states to low-dimensional subspaces in which these tasks are performed.

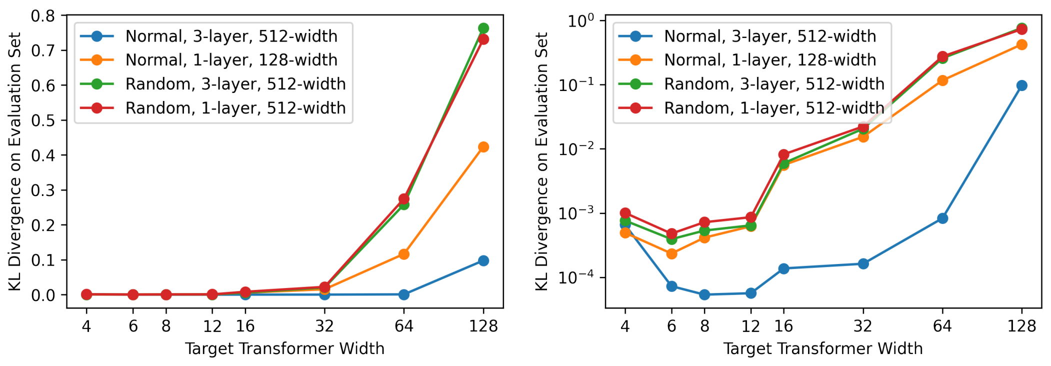


Figure 6: Kullback–Leibler divergence of circuit imitation with fully trained and random transformers (the lower the better). Both plots show the same set of results with different scales (linear and log) on the vertical axis.

However, our experiments have also highlighted several important differences between embedding-only and fully trained models, especially with regard to their parameter efficiency and information capacity. This paper leaves open the question of how computation in embedding-only models relates to fully trained ones—e.g. whether, during full training, the mechanisms we have discovered here evolve gradually into their fully trained forms, or whether fully trained models use entirely different pathways [10].

We anticipate that these random transformers will also provide an interesting new test-bed for interpretability research, and future work might investigate how learned feature codebooks [12, 6] and automated neuron labeling procedures [21, 5, 33] behave when applied to these models. Even more generally, these results motivate a closer study of the behavior of untrained models as a source of insight into differences between, and improvements upon, existing neural model architectures.

Acknowledgement

We would like to thank MIT SuperCloud for computational resources and Mingyang Deng for valuable discussions.

References

[1] Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Arhitectures and algorithms. *arXiv preprint arXiv:2401.12973*, 2024.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

[3] Ehsan Amid, Rohan Anil, Wojciech Kotłowski, and Manfred K Warmuth. Learning from randomly initialized neural network features. *arXiv preprint arXiv:2202.06438*, 2022.

[4] Anthropic. Long context prompting for Claude 2.1, 2023. URL `https://www.anthropic.com/news/claude-2-1-prompting`.

[5] Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models. *OpenAI*, 2023. URL `https://openai.com/index/language-models-can-explain-neurons-in-language-models/`.

[6] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language

models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Seventh International Conference on Learning Representations*, pages 1–17, 2019.

[9] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[10] Angelica Chen, Ravid Schwartz-Ziv, Kyunghyun Cho, Matthew L Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in mlms. *arXiv preprint arXiv:2309.07311*, 2023.

[11] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained BERT networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

[12] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

[13] 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. *arXiv preprint arXiv:2207.02098*, 2022.

[14] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[15] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. In *Seventh International Conference on Learning Representations*, 2019.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[17] Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

[18] Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

[19] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[20] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[21] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2021.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[23] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[24] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

[25] Yingcong Li, Muhammed Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In *International Conference on Machine Learning*, pages 19565–19594. PMLR, 2023.

[26] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[28] Cedric Lothritz, Kevin Allix, Lisa Veiber, Jacques Klein, and Tegawendé François D Assise Bissyande. Evaluating pretrained transformer-based models on the task of fine-grained named entity recognition. In *28th International Conference on Computational Linguistics*, 2020.

[29] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009.

[30] Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-Math: Unlocking the potential of SLMs in grade school math. *arXiv preprint arXiv:2402.14830*, 2024.

[31] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations*, 2023.

[32] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

[33] Tuomas Oikarinen and Tsui-Wei Weng. Clip-dissect: Automatic description of neuron representations in deep vision networks. In *International Conference on Learning Representations*, 2023.

[34] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[35] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[36] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[37] Philip Quirke and Fazl Barez. Understanding addition in transformers. *arXiv preprint arXiv:2310.13121*, 2023.

[38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[39] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11893–11902, 2020.

[40] Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *ICML*, volume 2, page 6, 2011.

[41] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2018.

[42] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pages 471–482, 2007.

[43] 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.

[44] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[45] Yun-Yun Tsai, Pin-Yu Chen, and Tsung-Yi Ho. Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources. In *International Conference on Machine Learning*, pages 9614–9624. PMLR, 2020.

[46] 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.

[47] Kaiyue Wen, Yuchen Li, Bingbin Liu, and Andrej Risteski. Transformers are uninterpretable with myopic methods: a case study with bounded dyck grammars. *Advances in Neural Information Processing Systems*, 36, 2024.

[48] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[49] Chao-Han Huck Yang, Yun-Yun Tsai, and Pin-Yu Chen. Voice2series: Reprogramming acoustic models for time series classification. In *International conference on machine learning*, pages 11808–11819. PMLR, 2021.

[50] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *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)*, pages 3770–3785, 2021.

[51] Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, and Songfang Huang. How well do large language models perform in arithmetic tasks? *arXiv preprint arXiv:2304.02015*, 2023.

[52] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

[53] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

Supplementary material

A Limitations

Even within the class of transformers, the space of architectural decisions (both around model size and implementation of attention mechanisms, normalization procedures, tokenization, etc.) is very large; our experiments in this paper generally characterize a small part of this phase space. It is thus possible that some of the described trends will change as models grow or differ in parameterization details. Outside Section 4, our experiments have focused on standard transformer models, and do not answer whether these trends hold in other related linear attention [9, 35] or state-space model [19] families. Our discussion in Section 6 focused only on linear subspaces and used principal component analysis as the primary tool, but it is also possible that subspaces or latent semantics appear non-linearly which is not addressed by our current analysis.

B Impact statement

We do not anticipate any ethical concerns associated with these results and we believe understanding AI systems is a crucial step in harnessing their power for good.

C Computational Resources

Roughly 154 GPU days of NVidia V100 were spent on this project.

D Setup details

D.1 Data Curation and Tokenization Details

D.1.1 Modular Addition

Fix modulus $p = 199$. We randomly shuffle all possible inputs (p^2 of them) perform a 95%: 5% for training and test set. The split is the same (generated with the same seed and procedure) for all runs across all architectures.

D.1.2 Dynamically Generated Tasks

For these tasks, we used a stream of training data and a heldout test set. The test set is fixed for all runs across all architectures. The training data is generated on the fly within the same distribution to avoid overfitting.

Needle-in-a-Haystack The number of entities (marker-value pairs) is first uniformly generated in $[1, 30]$. The values are generated as integers in $[1, 127]$. The markers are generated as distinct integers in $[127 + 1, 128 + 30]$. The final query token is the asked marker (uniformly chosen) plus 30. For example, `a 1 b 2 c 3 d 4` $\underline{\text{b}}$ in token ids would be $[128, 1, 129, 2, 130, 3, 131, 4, 159]$ and the expected output will be token 2.

Decimal Addition Fix the number of digits $l = 10$. The two numbers to be added are independently uniformly generated within $[10^9, 10^{10} - 1]$. The added numbers and the results are reversed. The addition sign has token id 10 and the equal sign has id 11. For example, $1111111112+2222222223 =$ in token ids would be $[2, 1, \cdots, 1, 10, 3, 2, \cdots, 2, 11]$. The result uses $[20, 29]$ to represent digits and 30 to signal the end of output. For the example, the expected output will be 3333333335 encoded as $[25, 23, \cdots, 23, 30]$.

Parentheses Balancing

The opening parenthesis has token id 1, the closing parenthesis has token id 2, and the question mark has token id 3. For the result, 2 signals balanced and 1 signals unbalanced. For example, `(())()?` in token ids will be $[1, 1, 2, 2, 1, 2, 3]$ and the expected output is balanced, token 2.

As sequences of uniformly random parentheses are (a) likely unbalanced (b) easier to validate, we performed a generate-then-mutate strategy to generate strong data for the task.

Generate With $1/3$ probability, we generate a random sequence of at most 60 parentheses (length uniformly chosen, then independently all the parentheses). With $2/3$ probability, we generate a balanced parentheses sequence. We first uniformly choose the number of pairs of parentheses t in $[1, 30]$. We then generate recursively: when $t \geq 2$, with $1/2$ probability we generate a $(t-1)$-parentheses sequence recursively and add a pair of parentheses on the outside, with $1/2$ probability we uniformly sample $u \in [1, t-1]$ and output the concatenation of a u-parentheses and a $(t-u)$-parentheses sequence generated recursively.

Mutate With $1/2$ probability, we choose some random index pairs and swap the corresponding parentheses. Independently with $1/2$ probability, we (then) choose some random indices and flip the corresponding parentheses. The number of pairs and indices are sampled according to a geometric distribution.

Circuit Imitation The inputs are uniformly random 40-token sequences of integer tokens $[0, 511]$. The target output is the distribution from a target transformer generated as in Appendix D.2.2 below. The width of the target transformer is chosen in $[4, 128]$ (Figure 6).

D.1.3 Memorization

For every integer pair $x \in [0, 511]$, $y \in [512, 512 + 511]$, we generate one data point with input $[x, y]$ and output z uniform in $[0, 511]$. There is no "test split" as the goal is to memorize all the associations.

D.1.4 Language Modeling

We used the original train-test split of the TinyStories dataset [14]. We trained a 10000-token BPE tokenizer from the training split alone.

D.2 Model Details

D.2.1 Transformers

We use the GPT-2 [38] implementation of Huggingface [48]. Dropout and weight tying are disabled. The activation function is kept as the default GeLU [20].

Specifically, all the weights of feed-forward layers are initialized by sampling from isotropic Gaussians with mean 0 and standard deviation $0.02/\sqrt{2n}$ where n in the number of layers. All the bias matrices are initialized with zeroes. All the other weight matrices (including key, value, query, embedding, unembedding matrices) are initialized with 0-mean Gaussians with standard deviation 0.02. The affine transformations in layer normalizations are initialized as identity.

We used two layer transformers except for the language modeling and circuit imitation task. The number of parameters for the algorithmic tasks can be found in Table 6.

Task	$E_{\textbf{token}}$	$E_{\textbf{pos}}$	U	**Intermediate Layers** F
Modular Addition	99,328	5,120	99,328	25,194,496
Needle-in-a-Haystack	262,144	102,400	262,144	25,194,496
Decimal Addition	31,744	40,960	31,744	25,194,496
Parenthesis Balancing	4,096	81,920	4,096	25,194,496

Table 6: Number of parameters of each type for the 2-layer 1024-width transformers in Section 4.

D.2.2 Target Transformer in Circuit Imitation

Based on the initialization in Appendix D.2.1, we make the following modifications.

Feed-forward Layers: Standard deviation scaled up by 20x: changed to $0.4/\sqrt{2n}$.

Task and Transformer Type		Principal Component Basis			Neuron Basis		
Task	**Model**	**Emb**	**L1**	**L2**	**Emb**	**L1**	**L2**
Decimal Addition	Normal	40.7%	91.6%	89.6%	3.3%	6.4%	6.8%
	Random	73.3%	62.6%	51.4%	4.3%	4.0%	3.6%
Needle-in-a-Haystack	Normal	31.4%	75.4%	42.5%	2.8%	5.6%	3.9%
	Random	51.3%	42.6%	33.9%	4.4%	4.6%	3.9%
Balanced Parentheses	Normal	47.5%	100.0%	99.9%	7.0%	11.1%	9.4%
	Random	85.2%	89.1%	84.7%	9.7%	8.0%	6.7%

Table 7: Median explained variance from top 10 directions under principal and neuron basis collected from width 512 transformers. Rounded to one decimal piece. See Table 4 for more details.

Attention Matrices (key, value, query): Standard deviation scaled up by 10x: changed to 0.4.

Unembedding Matrix: Standard deviation changed to $2/\sqrt{n}$.

After such modifications, we measured the entropy of output distribution on two random inputs. Across all the widths, the mean entropy stays within $[2.89, 3.02]$. We also measured similarity of output distributions on different inputs to prevent mode-collapse-like results, and the mean KL divergence of output distributions on random inputs is $[0.8, 3.3]$ across all widths.

D.2.3 LSTM

We used textbook long short-term memory [22] with an encoding layer and an unembedding (projection) layer added.

D.3 Training Details

For synthetic experiments, we used AdamW optimizer [27] with a learning rate 10^{-3} and weight decay 10^{-3}. For LSTM a learning rate 5×10^{-3} is used for faster convergence. For the language modeling task, we used AdamW optimizer with a learning rate 6×10^{-4} and weight decay 0.1. We clip all gradient norms at 1.

For random transformer experiments, the intermediate (query, embedding, unembedding, feed-forward) layers are kept as randomly initialized. We use a fixed number of training steps and no early stopping.

Modular Addition: 5000 epoches. Batch size 4000.

Needle-in-a-Haystack, Decimal Addition, Parentheses Balancing, Circuit Imitation: 10^4 steps of batch size 1000. Again, the training data is generated dynamically.

Memorization: 21000 epoches. Batch size 2^{15}.

Language Modeling: 5 epoches. Batch size 20 and context window 512.

E Performance on synthetic tasks across model scales

In Fig. 7 we provide the test accuracy of needle-in-a-haystack, decimal addition and parenthese balancing for fully trained and random transformers across different model widths. Note that the 1024-width fully trained transformer had trouble reaching perfect accuracy in parentheses balancing, likely due to imperfect hyperparameter choices.

We also include the explained variance measurements on 512-width models in these three tasks for completeness (Table 7). Generally more variances are explained from the top directions as the models are narrower.

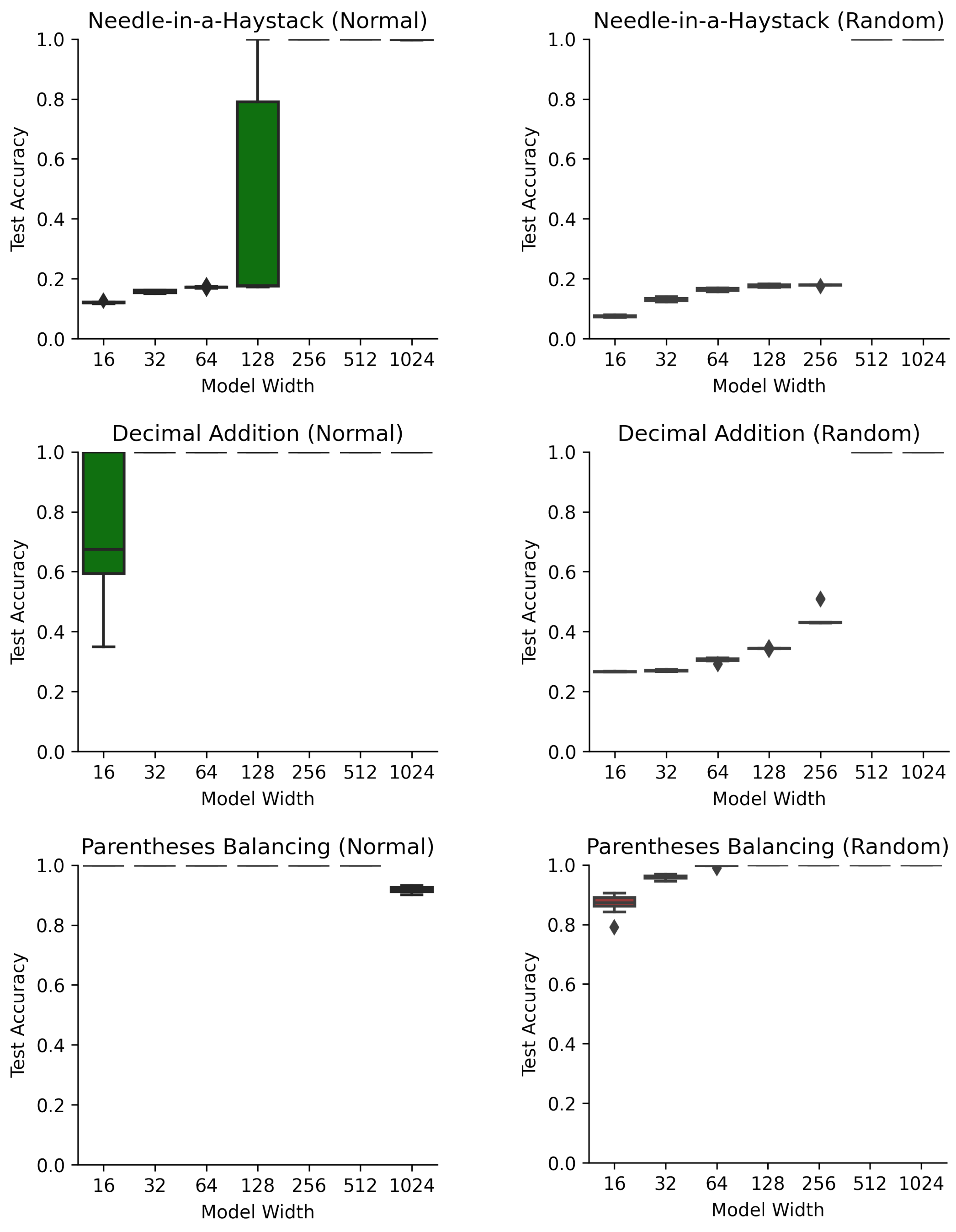


Figure 7: Box plot of test accuracy of synthetic tasks across model scales.

F Constant-dimensional subspaces are sufficient for many synthetic tasks

Definition F.1 (Intermediate complexity of tasks (informal))**.** For a family of neural networks with varying intermediate width, we define the **intermediate complexity** of a task as the minimal width of *intermediate* layers required to succeed in the task.

For example, a 2-layer transformer with intermediate width 2 starts by embed the tokens into the 2-dimensional hidden space, passes them through two transformer blocks of width 2, then linearly project the activation for the final output (unembedding). If one task could be solved with one such transformer, its intermediate complexity for 2-layer transformers would be at most 2.

If we consider the activations as the working memory of the LLM, intuitively a task has low intermediate complexity if it requires a low working memory per token. In the following, we show that all the synthetic tasks we proposed indeed have constant intermediate complexity for 2-layer transformers, thus they only require a constant-dimensional intermediate subspaces to work on.

Modular Addition We may use a transformer to implement three instances of the *Pizza* algorithm described in Zhong et al. [52] (multiple copies are needed to avoid the "antipodal pairs" problem described there). Each implementation of this algorithm requires intermediate representations of size 2, so the intermediate complexity for 2-layer transformers is ≤ 6.

Needle-in-a-Haystack We may use a transformer to implement induction heads [34] as follows: We first store each token's preceding token in its activations by attending to the previous position. We then retrieve the answer by attending to the position storing the matching token in the previous step. Both steps could be done by using attention to compute squares of differences and take the minimum. The number of hidden states needed to distinguish each symbol scales only with the size of the vocabulary, and is at most equal to the vocabulary size, so the intermediate complexity is at most twice the vobaulary size.

Parentheses Balancing Let the given sequence of parentheses be $p_1, p_2, \cdots, p_n$. Let $x_i = \sum_{j=1}^{i} c_i$ where $c_i = 1$ if p_i is an open parenthesis and $c_i = -1$ if p_i is a closing parenthesis. The given sequence is a valid parentheses sequence if and only if $x_n = 0$ and $\min_{i=1}^{n} x_i = 0$. In a transformer, we can first compute x_i/i with a uniform attention. Then, we attend to the position with minimum x_i/i, breaking ties by position (let the attention on i pre-softmax be $T(-x_i/i + \epsilon \cdot i)$ for large enough T and small enough $\epsilon > 0$). For valid parentheses sequences, the last position should only attend to itself. We then check if the attended position is indeed the last one and has $x_i/i = 0$ by computing squares of differences. The intermediate complexity for 2-layer transformers is thereby again constant. (note that a different construction for bounded-depth Dyck was given in [50])

Decimal Addition

Let the reversed list of digits of two numbers and their sum be $a_1, a_2, \cdots, a_n$, $b_1, b_2, \cdots, b_n$, and $c_1, c_2, \cdots$. Let $x_i = \sum_{j=1}^{i} a_j 10^{j-i}$, $y_i = \sum_{j=1}^{i} b_j 10^{j-i}$, $z_i = \sum_{j=1}^{i-1} c_j 10^{j-i}$, we have $c_i = (x_i + y_i - z_i) \bmod 10$: $a + b \equiv \sum_{j=1}^{i} 10^{j-1}(a_j + b_j) \pmod{10^i}$, $(a+b) \bmod 10^{i-1} = \sum_{j=1}^{i-1} c_j 10^{j-1}$, and $c_i = ((a+b) \bmod 10^i - (a+b) \bmod 10^{i-1})/10^{i-1}$.

Prepare: Take small $\epsilon > 0$. For positions $1, 2, \cdots, n$, let the positional embedding of position i be $[1, 10^i, 10^{-i}\epsilon]$. In the first head of the first layer, take the first dimension in the positional embedding for both Q and K, so we get a uniform attention on the prefix from which $\alpha_i = \frac{1}{i}\sum_{j=1}^{i} a_j$ can be calculated. In the second head of the first layer, take the third dimension in the positional embedding for Q and the second dimension in the positional embedding for K, so the contribution from the j-th position to the i-th position is $\frac{1}{Z_i} e^{\epsilon 10^{j-i}} \approx \epsilon \frac{1}{Z_i}(10^{j-i} + 1)$ ($e^c \approx 1 + c$ for $0 < c \ll 1$) for normalizing constant Z_i, we can then calculate $\beta_i = \epsilon \frac{1}{Z_i} \sum_{j=1}^{i} a_j (10^{j-i} + 1)$. We then have $x_i = \frac{Z_i}{\epsilon}\beta_i - i\alpha_i$ which we will utilize in the next step. Similarly, we can have y_i and z_i ready in the corresponding position (position of b_i and c_{i-1}).

Generate $x_i + y_i - z_i$: In the second layer, attend from the position of a_i and b_i at the position of c_{i-1}. Specifically, set the attention from the a_k's position to the c_{i-1}'s position be $\epsilon i - \Lambda \cos((k - i)/n) = \epsilon i - \Lambda(\cos(k/n)\cos(i/n) + \sin(k/n)\sin(i/n))$ pre-softmax. This could be done using three dimensions in the positional embeddings. We also set the attention from the plus sign to the c_{i-1}'s position be 1 pre-softmax. The attention from a_k to c_{i-1} will be negligible if $k \neq i$ and will be proportional to ϵi post-softmax for $k = i$. We can then calculate $-i\alpha_i$ from it and similarly $\frac{Z_i}{\epsilon}\beta_i$, and thus x_i, y_i and z_i.

Approximate the Answer: As $x_i + y_i - z_i$ is an integer from $[0, 19]$ and we can also compute an affine transform of it, we may now simply proceed with the universal approximation theorem. As an example, in ReLU networks we may check if $x_i + y_i - z_i = v$ by noticing

$$\text{ReLU}(x_i+y_i-z_i-(v+1))+\text{ReLU}(x_i+y_i-z_i-(v-1))-2\text{ReLU}(x_i+y_i-z_i-v) = [x_i+y_i-z_i = v],$$

So to create an indicator variable $[x_i + y_i - z_i = 1]$ we simply need to generate $x_i + y_i - z_i$, $x_i + y_i - z_i - 1$, $x_i + y_i - z_i - 2$, pre-ReLU.

The intermediate complexity for 2-layer transformers is thereby again constant. This approach is quite crude so it is likely that the constant can be greatly improved.

G Difficulty of the synthetic tasks

Chomsky hierarchy has been found to be predictive of the neural architectures' performances, and especially length-generalization performances, in algorithmic setups [13]. We list the Chomsky hierarchy of the synthetic tasks we picked in Table 8.

Task	**Chomsky Hierarchy**
Needle-in-a-Haystack	Regular
Decimal Addition	Context Sensitive
Parenthesis Balancing	Context Free

Table 8: Classification of tasks according to the Chomsky hierarchy. For the decimal addition task, we consider the most flexible setting where all the strings representing decimal additions (does not have to be equal-lengthed) are considered within the language.

We also examined a baseline with linearized transformer. In this variant of transformer, the attention matrix (softmax(QK^T)) is replaced with a lower diagonal matrix of 1s. In other words, the attention is replaced by a simple token prefix sum mechanism. We tested such transformers with width 512 and 2 layers on the synthetic tasks as a baseline performance. The result is shown in Table 9. We can see that such transformers have large performance gaps with normal transformers, confirming the difficulty of our chosen tasks.

Task	**Accuracy (%)**
Needle-in-a-haystack	17.67
Decimal Addition	26.39
Parenthesis Balancing	97.32

Table 9: Linearized transformer performance in terms of accuracy.

H Accuracy curve of the memorization task

The accuracy curve in the memorization task during training is shown in Fig. 8. Training of both transformers has converged.

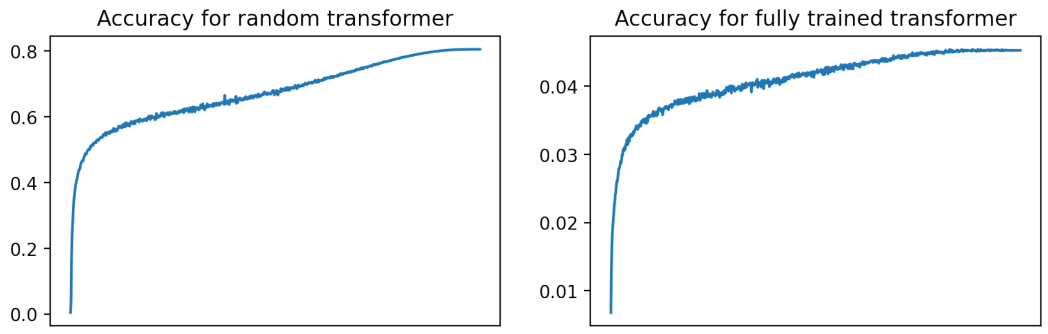


Figure 8: During training, the accuracy curve from fully trained and random transformers in the memorization task (Section 5.1). Note that the evaluation set is exactly the training set as the goal is to memorize.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All our claims are based on empirical evidences.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Appendix A.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The only theoretical result is included in Appendix F where the full proof is given.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We tried our best to convey all the experiment details and we will also be releasing code and data after some final cleanup.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We plan to release code and data after finalizing.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details are included in Appendix D.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We include the 80% CI bars in Appendix E. Some experiments are not repeated due to computational constraints.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We reviewed the Code of Ethics and we believe our work will not cause potential harms and negative societal impacts.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Appendix B.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our results do not pose high risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cited the original papers for each asset.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We will release the code and trained models as well as documentation on how to apply them.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing or human-subject research is involved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No crowdsourcing or human-subject research is involved.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.