THE MEMORIZATION CAPABILITIES OF NEURAL NETWORKS: UNDERSTANDING SCALING LAWS IN AUTOREGRESSIVE AND SEQUENCE-TO-SEQUENCE LEARNING

Niranjan Vijaya Krishnan, Christine Guo, Diya Hundiwala
Princeton University
{nv2608, cg4302, dh3163}@princeton.edu

ABSTRACT

We investigate the memorization capacity of multi-layer perceptrons (MLPs) and decoder-only transformers on synthetic random-mapping tasks. Given a fixed vocabulary, sequence length, and total number of model parameters, we examine how varying different model features—such as depth, width, and attention mechanisms—affects the model's ability to memorize. The memorization threshold is defined as the maximum number of random sequence mappings the model can learn with perfect accuracy. MLPs are applied to sequence-to-sequence learning, and decoder-only transformers are tasked with autoregressive learning.

Our results show that MLPs leverage width, while transformers depend on multi-head attention to improve memorization. We find that MLPs achieve a much higher memorization threshold under a similar number of total parameters.

1 INTRODUCTION

In this project, we explore the memorization capacity of neural architectures, specifically, transformers and MLPs, on synthetic random mappings. We focus on autoregressive and sequence-to-sequence tasks with fixed-size vocabularies and random bijective mappings to study pure memorization. Our goal is to understand how architectural parameters like width, depth, and attention affect a model's ability to memorize random mappings and how memorization scales with model size.

2 RELATED WORKS

Understanding the memorization capacity of neural networks has become an area of focus within deep learning research, revealing both the limits of model expressivity and also potential risks such as overfitting and privacy leakage.

2.1 WHY MEMORIZATION MATTERS

Early work has highlighted that over-parameterized models, such as large transformers and MLPs, can memorize substantial amounts of training data, in some cases to the point of perfect interpolation on random labels. This sparks concerns in overfitting, where memorization would undermine generalization to unseen data Zhang et al. (2017), and privacy leakage, where memorized sensitive data could be exposed Carlini et al. (2021).

Despite these drawbacks, memorization is surprisingly not always detrimental. Feldman (2020) showed that memorization can enhance generalization in settings with long-tailed data distributions, where rare examples are vital to model performance. Furthermore, studies of double descent, such as Belkin et al. (2019) and Nakkiran et al. (2020), revealed that models that cross the interpolation threshold, memorizing all training points, would often generalize better with further overparameterization.

2.2 INFORMATION-THEORETIC PERSPECTIVES

Recent work provides deeper insight into how many bits of information a network can store. Park et al. (2021) and Vardi & Shamir (2022) demonstrated that depth dramatically boosts memorization, showing that deep MLPs can memorize N examples using only $O(N^{2/3})$ or even $\tilde{O}(\sqrt{N})$ parameters, given that the weights have high precision. This introduces a bit-complexity tradeoff: a network's memorization capacity depends on both the number of parameters and the number of bits per parameter.

In terms of transformers, Kim et al. (2023) extended upon these findings, proving that transformers can memorize N sequence mappings using around $O(d + n + \sqrt{nN})$ parameters, given that d is the embedding size and n is the sequence length. Empirical work has shown that these theoretical predictions align closely with real transformer behavior, suggesting that transformers are highly parameter-efficient memorization machines.

2.3 THEORETICAL BOUNDS OF TRANSFORMER MEMORIZATION CAPACITY

Mahdavi et al. (2024) theoretically investigated the role of attention heads and how they affect the number of sequences the Transformer can memorize. They found that an attention layer with H heads, dimension d, and context size $n < d$, with $\Theta(Hd^2)$ parameters, can memorize $\Omega(Hn)$ examples. Mahdavi further finds that each attention head memorizes a different subset of example sequences, dividing memorization capacity among attention heads. This implies that increasing the number of attention heads linearly increases the memorization capacity of a Transformer.

On the other hand, Kajitsuka & Sato (2025) establish a theoretical lower bound for the memorization capacity of Transformers. They find that in the sequence-to-sequence setting, $\widetilde{O}(\sqrt{nN})$ parameters are sufficient for memorizing N input sequences of length n. However in the next-token prediction setting, $\widetilde{O}(\sqrt{N})$ parameters are needed. This provides a useful reference to assess how efficiently a Transformer utilizes its parameter constraint to memorize sequences.

2.4 EMPIRICAL EVIDENCE AND PRACTICAL IMPLICATIONS

Arpit et al. (2017) found that neural networks first learn patterns and only later memorize exceptions/noise, while Power et al. (2022) explored grokking, where memorization unexpectedly transitions into true generalization during extended training. These results highlight that memorization is dynamic and staged, rather than uniform.

Lu et al. (2024) studied the knowledge capacity of LLMs and found that the knowledge capacity of an LLM has a linear and negative exponential law relationship with model size and training epochs. This suggests that trained Transformers may not necessarily be memorizing training data but also generalizing unseen data.

Memorization raises serious privacy concerns in practice. Carlini et al. (2021; 2023) demonstrated that LLMs like GPT-2 can reproduce verbatim training data, and that memorization risk scales with model size and training data redundancy. This has motivated further research into privacy-preserving training methods.

We situate our work within this literature by focusing on controlled synthetic random mapping tasks to isolate pure memorization capacity. Unlike studies that focus on naturalistic data, our setup removes semantic patterns, enabling a clear measurement of the memorization threshold as a function of model size, depth, and architecture type. By comparing MLPs and transformers, we extend upon prior work from Park et al. (2021) and Kim et al. (2023), empirically testing scaling predictions in sequence-to-sequence tasks.

3 METHODOLOGY

3.1 MODEL ARCHITECTURES & LEARNING TASKS

The two model architectures explored are multi-layer perceptrons (MLPs) and decoder-only transformers. The following explains each model's architectural features and the task each one analyzes.

3.1.1 MULTI-LAYER PERCEPTRONS (MLPS) & SEQ2SEQ

Although MLPs are not typically described as sequence learning models, this task can be interpreted as a form of sequence-to-sequence learning, where the goal is to map one fixed-length sequence to another. We chose to analyze MLPs because their architecture makes them inherently well-suited for dictionary tasks. MLPs do not take into account any sequential dependencies or semantic meaning (unlike transformers); instead, they treat the input as one vector and apply a series of nonlinear transformations to learn the expected output. This makes them effective at learning random mappings such as our task. Additionally, they can learn to sparsely activate certain neurons, boosting their accuracy in memorization tasks.

The MLP has three parameters, the number of hidden layers (*num_layers*), the dimension of the hidden layer (*hidden_size*), as well as the dimension of the input and output layers (these are always equal and are d_model_{MLP}). The input to the MLP is a flattened, one-hot vector representing a sequence of 10 tokens, where our vocabulary size is 10. Therefore, d_model_{MLP} is held constant where $d\_model_{MLP} = |V| * seq\_len = 10 * 10 = 100$.

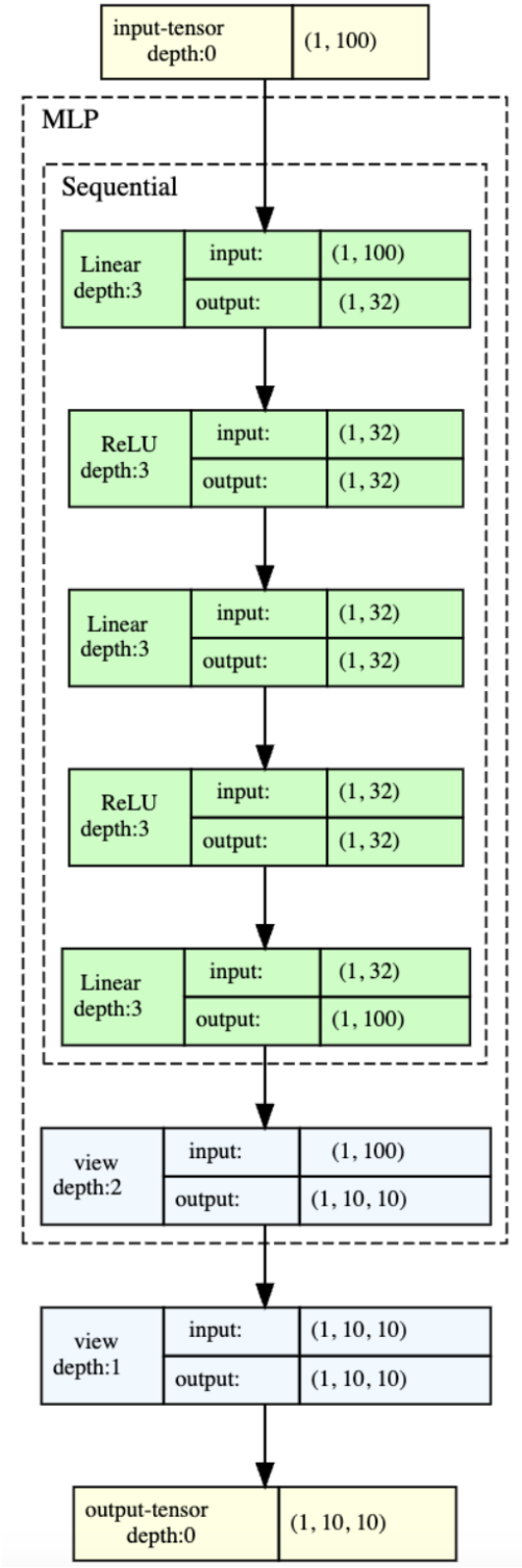

Figure 1: Visualization of the MLP (Multi-Layer Perceptron) architecture. The model takes a flattened one-hot input of size $(\text{batch\_size}, \text{seq\_len} \times \text{vocab\_size})$ and passes it through multiple fully connected layers with ReLU activations. The final output is reshaped back into $(\text{batch\_size}, \text{seq\_len}, \text{vocab\_size})$ logits. This simple feedforward network is used as a baseline for memorization capacity experiments.

3.1.2 DECODER-ONLY TRANSFORMERS & AUTOREGRESSIVE LEARNING

We only analyze decoder-only transformers (as opposed to encoder-decoder transformers) to isolate and better understand the behavior of the decoder. The encoder introduces additional parameters to process and embed the input sequence, which can obscure the effects of core architectural features like depth and attention within the decoder itself. Therefore, by using a decoder-only model, we limit the task to autoregressive learning, where the model predicts the next token using the previously generated tokens. Using causal attention, model learns to predict the next token in a sequence without future context, making it easier to analyze the contributions of key architectural features.

However, it is important to note that we do choose to embed our one-hot vectors in the token embedding dimension d_model_{TRF} before passing it as an input to the transformer. This was done for multiple reasons. In most literature, d_{model}, the dimension of the input tokens, is equivalent to d_{attn}, the attention output. Therefore, without this embedding step, $d\_model_{TRF} = |V| = 10 = d_{attn}$, significantly limiting the expressive capabilities of our transformer. Empirically, we found that the results for when $d_{model} = d_{attn}$ are quite poor. While we debated letting d_{attn} be different, we decided against it in order to maintain consistency when comparing to previous literature.

Our decoder-only transformer has 4 parameters: the token embedding dimension (d_model_{TRF}), the number of attention heads (*num_heads*), the number of layers (*num_layers*), the feedforward layer dimension of the MLP (d_{ff}).

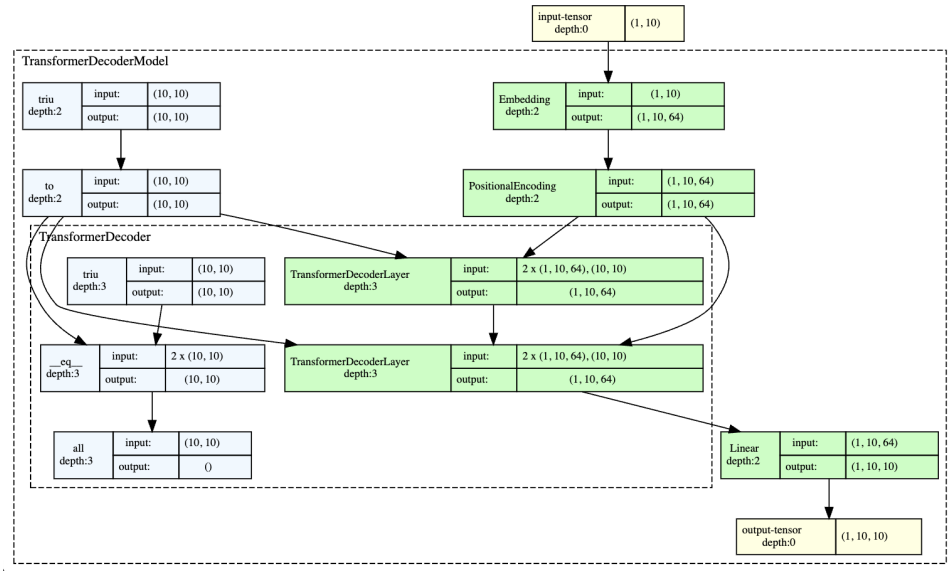

Figure 2: Visualization of the Transformer Decoder model used in our experiments. The model consists of an embedding layer, positional encoding, and multiple stacked `TransformerDecoderLayer` blocks. The architecture processes input sequences of token indices with autoregressive masking (causal mask) and outputs token-level logits for each position. Green blocks represent learnable layers, while blue blocks show operations like mask generation and tensor reshaping.

3.2 DATASET GENERATION

Given a dataset size *subset_size*, we generate synthetic datasets with fixed sequence length $seq\_len = 10$ and vocabulary $V = \{0, 1, \ldots, |V| - 1\}$ where the vocabulary size $|V| = 10$. The dataset size is equivalent to the number of generated mappings. To generate each sequence, we randomly choose *seq_len* tokens from the vocabulary. Once all of the input sequences are generated, we check that there are no repeated sequences within this set to ensure a bijective mapping. The same is done for all label sequences. Finally, we randomize the orderings of the label sequences to ensure a random (un-learnable) mapping function from input to label sequences. This is done because the mapping is done index-wise between the sets of sequences.

3.3 THRESHOLD SEARCH

In order to determine the memorization threshold–the maximum number of mappings a given model can memorize–we perform a binary search on the entire search space of all possible mappings of size S_{low} to S_{high}, where S_{high} is a known upper bound that the model cannot memorize and S_{low} is a known lower bound that the model can memorize.

Memorization failure occurs if, on a given epoch, perfect accuracy is not achieved and any of the following: 1) the maximum number of epochs (500) was reached or 2) validation loss plateaus for the last three epochs (plateau is defined as no improvement by at least 0.0001 for the last 3 epochs). We keep track of whether the maximum number of epochs was reached using a flag and re-run the data on a higher epoch count if necessary.

Important to note is that the training, validation, and test datasets are all the same because we are examining memorization and not generalization.

In order to determine S_{low}, S_{high}, we do the following bounds search:

1. Initialize S_{high} as 1.
2. Train model on $subset\_size = S_{high}$ random mappings.
3. If model achieves perfect accuracy, multiply S_{high} by a factor of 2 and repeat steps 2-3. If model fails, set $S_{low} = S_{high}/2$ and end search.

Once the bounds are determined, we apply a binary search to identify the exact memorization threshold:

1. Initialize S_{low}, S_{high} as dataset size bounds.
2. Train the model on $subset\_size = (S_{low} + S_{high})//2$ random mappings.
3. If accuracy $< 100\%$, lower S_{high}; if accuracy = 100%, raise S_{low}.
4. Repeat steps 2-3 until convergence.

We repeat runs to account for noisy results. Models are trained using the Hugging Face Trainer with AdamW optimizer, batch size of 32, and weight decay of 0.01. For the MLP, we use a learning rate of 1e-2, while 5e-4 is used for the transformer. These parameters were decided after thorough experimentation.

3.4 METHODOLOGICAL SHIFT: FROM INITIAL APPROACH TO CURRENT DESIGN

Since our presentation, we've changed our approach to various methodology features: model architecture, dataset generation, and threshold search.

Model Architecture: Our previous architecture kept all model parameters constant while only varying one of them. However, the analysis was less interesting because the total number of model parameters was increasing which is known to be strongly correlated with model performance. Now, we keep the total number of parameters constant while varying the relationship between pairs of parameters–such as number of layers and the dimension of the feedforward layer in the MLP.

Dataset Generation: Previously, we divided the vocabulary into input and label tokens, where only input tokens could be used to generated input sequences and vice versa. However, this made our notation and definition of memorization more complex. Therefore, we have expanded to a unified vocabulary approach for input and label sequences.

Threshold Search: Before, our binary search bounds were related to the size of the vocabulary our model was trained on. Therefore, on each iteration of the search, a new vocabulary and sequence mappings were generated. To maintain consistency and simplify our approach, we now keep the vocabulary size constant and much smaller, with a longer sequence length.

3.5 Parameter Selection

3.5.1 Multi-Layer Perceptrons (MLPs)

In order to determine the parameter settings for *hidden_size* and *num_layers*, we solve for all possible *hidden_size* for all *num_layers* $\in \{0, 1, 2, 3, 4\}$ while maintaining a constant total number of parameters. Let N be the number of total parameters, d_{hs} = *hidden_size*, $d_{model} = d\_model_{MLP}$, $L = num\_layers$:

$$N = \begin{cases} d_{model}^2 + d_{model} & \text{if } L = 0 \\ 2d_{model}d_{hs} + (L-1)d_{hs}^2 + Ld_{hs} + d_{model} & \text{if } L > 0 \end{cases}$$

Using the python library scipy.optimize, for a given N, we solve the equation for all *num_layers* $\in \{0, 1, 2, 3, 4\}$ to find the correct *hidden_size* that maintains that constant number of parameters. We repeat this for all $N \in \{10000, 30000, 90000\}$. The scaling of N was decided using the paper Kim et al. (2023); Vardi & Shamir (2022) as reference.

3.5.2 Transformer

To measure how different transformer architectural parameters affect memorization capacity, we train numerous transformer models with varying parameters. Each trained Transformer model varies the number of layers, hidden size, feedforward dimension, and number of attention heads, while establishing a parameter constraint that the total number of parameters is approximately 10,000. This allows us to compare how each model efficiently uses its fixed number of model parameters to memorize sequences.

We vary transformer architecture across a series of models by adjusting model depth and width. We define depth as the number of layers n_{layers} and width as the hidden size d_{model}, feedforward dimension d_{ff}, and attention output size d_{attn}. To keep the total number of parameters constant, we increase depth while appropriately decreasing width at the correct scale.

There are several ways to decrease the width. Prior literature decrease the width while maintaining the following ratio: $d_{\text{model}} = d_{\text{attn}} = d_{\text{ff}}/4$ Vaswani et al. (2017); Devlin et al. (2019); Kaplan et al. (2020); Hoffmann et al. (2022). However, we adopt the design introduced by Petty et al. (2024) where we vary d_{ff} while holding $d_{\text{model}} = d_{\text{attn}}$ constant. This way, any differences in Transformer memorization capacity is truly due to depth and not any attention mechanisms.

We identify model parameters by the following method. We start with initial values of $(n^0_{\text{layers}}, d^0_{\text{ff}})$ that satisfy $N(n^0_{\text{layers}}, d^0_{\text{ff}}) = p$ where p is the parameter constraint and $N(n_{\text{layers}}, d_{\text{ff}})$ is the total number of parameters. We find values $l, f(l)$ such that $N(n^0_{\text{layers}}, d^0_{\text{ff}}) = (n^0_{\text{layers}} + l, d^0_{\text{ff}} - f(l))$. As a result, if we add l layers, we decrease the the feedforward dimension by $f(l)$. We round the calculated value of $f(l)$ accordingly to give d_{ff} an integer value and train a model with the updated parameters.

4 Results

We report memorization thresholds as a function of hidden size, depth, and attention, comparing MLPs and transformers. We analyze width vs. depth trade-offs and the role of attention in sequence memorization without semantics Park et al. (2021); Vardi & Shamir (2022); Kim et al. (2023).

5 MLP Results and Insights

We evaluate the memorization capacity of multi-layer perceptrons (MLPs) across three different total parameter budgets: 10,000, 30,000, and 90,000 parameters. For each budget, we systematically vary the hidden size and number of layers to study how width and depth trade off under a fixed parameter constraint.

5.1 Observed Trends Across Parameter Budgets

Our results (Tables 1–3) show several consistent patterns:

- Single-layer MLPs (Depth = 1) consistently achieve the highest memorization threshold within each parameter budget. For example, at 10,000 parameters, an MLP of depth 1 reaches a mean threshold of 467, while deeper models with the same budget achieve lower thresholds.
- Adding depth, or more layers, reduces memorization capacity when the total parameter count is held constant. For example at 30,000 parameters, the threshold drops from 1728 with 1 layer to just 118 with 4 layers. Similarly, at 90,000 parameters, there is a drop from 3648 with a single layer to only 1 with over 4 layers.
- Models with zero hidden layers (Depth = 0) end up plateauing at a threshold of 224 across all parameter budgets, which is expected behavior as these models behave like a direct linear map with no hidden transformation capacity.

5.2 Width vs. Depth Tradeoff

These results suggest that width plays a dominant role in MLP memorization when parameter count is fixed. Increasing the number of layers (depth) requires a corresponding reduction in hidden size (width) to stay within the parameter budget, which sharply reduces memorization capacity. This aligns with theoretical work (e.g., Park et al. (2021); Vardi & Shamir (2022)) showing that width is critical for memorization in MLPs, and that depth alone cannot compensate if the width shrinks too much.

Interestingly, the drop-off is nonlinear: while increasing from 1 to 2 layers preserves some memorization (e.g., 995 at 30k params), further depth increases lead to steep declines (e.g., 256 at 3 layers, 118 at 4 layers). This suggests a sweet spot where shallow depth might offer minor gains (e.g., through additional expressivity) before the penalty of reduced width dominates.

5.3 Unexpected Results

One intriguing result is the increased variance at deeper settings. For instance, at 10k parameters, the 4-layer MLP shows a large standard deviation (41.2), reflecting unstable or inconsistent memorization performance across runs. This could be due to optimization challenges: deeper MLPs may suffer from vanishing gradients or difficulty converging when width is extremely constrained Arpit et al. (2017); Zhang et al. (2017). We were not able to explore this fully due to compute and time constraints, but this would be an interesting area of exploration.

Another notable observation is that even large parameter budgets (e.g., 90k) do not save deep MLPs from collapse in memorization capacity. At 5 layers, the memorization threshold remains extremely low (1 sequence memorized), highlighting that overparameterization alone cannot overcome the bottleneck created by low width.

5.4 Comparison to Prior Literature

These empirical findings align with theoretical predictions (e.g., Vardi & Shamir (2022)) that depth can amplify memorization only when paired with sufficient width. The sharp performance collapse at high depth suggests that, under a fixed parameter budget, width is the dominant factor for memorization on sequence-to-sequence tasks.

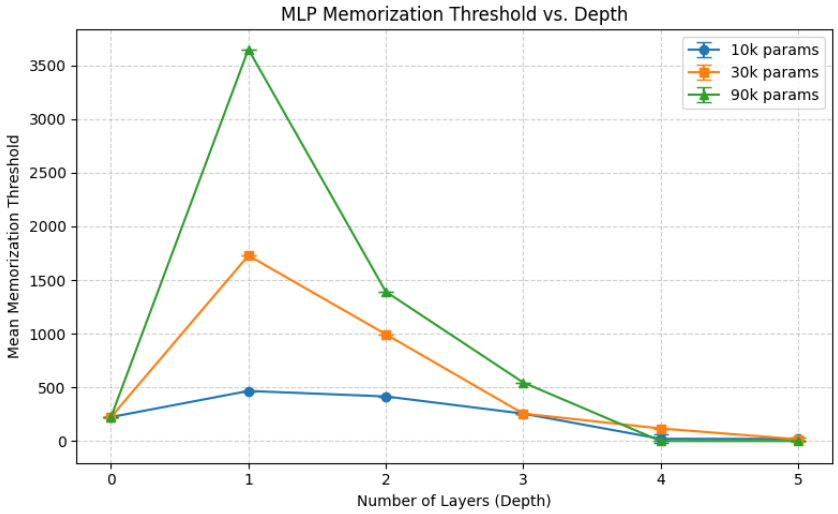

Figure 3: Memorization threshold vs. depth for MLPs across different parameter budgets (10k, 30k, 90k). Each line shows that as depth increases, memorization capacity sharply decreases, with depth-1 models consistently achieving the highest thresholds. This figure highlights the width vs. depth trade-off: deeper models require reduced hidden size to stay within a fixed parameter budget, which limits their memorization ability.

Our results also reinforce prior work (e.g., Arpit et al. (2017)) showing that MLPs memorize effectively when the architecture can create a large enough expressive capacity (wide layers), while purely adding depth without adequate width does not meaningfully improve memorization.

In summary, MLPs excel at memorization when their architecture allows for large hidden layers, even if depth remains shallow. This reflects the fundamental limitation of MLPs for memorization under tight parameter budgets: depth alone is insufficient without ample width.

Table 1: MLP Results (10,000 parameters)

hidden_size	**num_layers**	**Mean Threshold**	**Median Threshold**	**Std. Dev.**
0	0	224	224	0
49	1	467	467	0
41	2	416	416	0
36	3	256	256	0
33	4	21.6	1	41.2
30	5	20.6	33	15.19

Table 2: MLP Results (30,000 parameters)

hidden_size	**num_layers**	**Mean Threshold**	**Median Threshold**	**Std. Dev.**
0	0	224	224	0
149	1	1728	1728	0
99	2	995	995	0
82	3	256	256	0
71	4	118	118	0
65	5	16.8	2	18.13

Table 3: MLP Results (90,000 parameters)

hidden_size	num_layers	Mean Threshold	Median Threshold	Std. Dev.
0	0	224	224	0
447	1	3648	3648	0
215	2	1392	1392	0
167	3	546	546	0
142	4	1	1	0
126	5	1	1	0

6 Transformer Results and Insights

We evaluate the memorization capacity of transformer architectures across three different total parameter counts: 32,000, 64,000, and 128,000 parameters. For each total, we systematically vary the feedforward dimension and and number of layers, while keeping the hidden size constant (modeling the work of Petty et al. (2024)). This is done in order to maintain the target parameter count. Moreover, because the number of attention heads does not affect the total parameter count, we experimented with a wide variety such that the hidden size was divisible by the number of heads.

6.1 Observed Trends Across Parameter Budgets

Our results (Tables 4–6) reveal several consistent trends:

- Within each total parameter count, increasing the number of attention heads at fixed depth generally increases memorization thresholds. For instance, at 64,000 parameters and a depth 1 (*num_layers* = 1), memorization improves from 18 mappings (1 head) to 77 mappings (32 heads).
- Increasing depth from 1 to 2 or 3 layers leads to a modest reduction in memorization threshold, but the decline is not as sharp as in MLPs. At 128,000 parameters, a 2-layer transformer with 32 heads reaches a threshold of 32, compared to 31.2 at 1 layer—demonstrating relative robustness to depth.
- To stay within the total parameter count, deeper models have more narrow feedforward layers ($d_{ff} = 175$ for 3 layers vs. $d_{ff} = 1550$ for 1 layer at 128k params), but transformers maintain a substantive memorization capacity even with a more narrow width, unlike MLPs which fail to maintain a baseline threshold.

6.2 Attention, Width, and Depth Tradeoff

A key feature in transformers is that they benefit from multi-head attention in longer sequences. With a sequence length of 10, this is demonstrated as the average memorization threshold increases from 19.4 to 64 for a model with 1 compared to 16 attention heads (where $d_{ff} = 256$, $num\_layers = 2$, $hidden\_dim = 64$ as shown in Table 5). While the width nor depth of the model doesn't necessarily change, we see how attention heads enable greater expressivity and memorization capabilities in transformers. We were somewhat surprised by this result because our generated sequences do not contain any semantic meaning, therefore there isn't necessarily a specific feature that we expect the heads to learn. Moreover, the embedding dimension of the heads significantly reduce: for 1 total head, it has a dimension of 64, while for 16 heads, each one has a dimension of only 4. Although each head's expressivity is limited when there are more heads in the model, this is a worthy tradeoff for performance.

As for a direct comparison between width and depth, we see that increasing depth (*num_layers*) while reducing width (d_{ff}) does not have a significant difference when compared to decreasing depth (*num_layers*) and increasing width (d_{ff}). This is shown in Table 6.4 for a total of 180k parameters, where the average threshold changes by 2 between a model with $d_{ff} = 1550, num\_layers = 1$ versus a model with $d_{ff} = 512, num\_layers = 2$. Similarly in a model with 64k total parameters, moving from 1 layer (77 threshold at 32 heads) to 3 layers (41 threshold at 32 heads) still retains a significant portion of memorization.

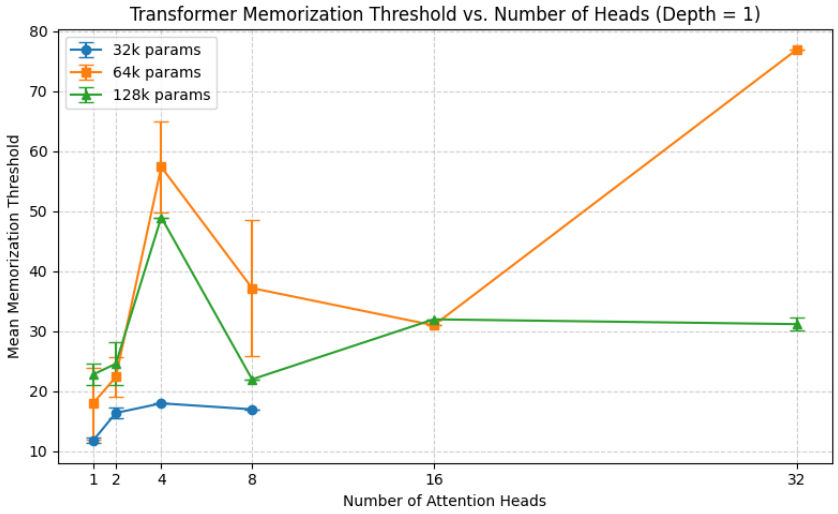

Figure 4: Memorization threshold vs. number of attention heads for transformers (fixed depth = 1) across different parameter budgets (32k, 64k, 128k). The results show a roughly linear scaling trend, with higher head counts consistently yielding better memorization capacity.

6.3 Unexpected Results

There were a few results that surprised us:

- In some cases (such as in a model with 128k params, depth 1), increasing heads beyond 32 does not boost memorization any further. In fact, performance often plateaus or dips (as seen in models with 32 heads vs. 64 heads which have thresholds of 31.2 and 32, respectively). This suggests possibly diminishing returns when splitting attention heads too finely.
- We observe larger standard deviations in deeper models. For example, the standard deviation is 18.12 at 32 heads, 3 layers, 128k parameters, which may reflect optimization challenges or sensitivity to initialization when both depth and head count are high.
- When comparing the number of total parameters, as seen in Figure 4, we witness how larger models (ones with higher total parameter counts) do not necessarily correlate with better performance. It seems that a model with 64k parameters performs better than that of with 128k parameters, especially with an increased number of attention heads.

6.4 Comparison to MLPs and Prior Literature

Transformers demonstrate a fundamentally different scaling pattern than MLPs. While MLPs rely heavily on hidden size for memorization (with a heavy impact on performance as depth increases), transformers show resilience: attention heads can help compensate for narrower d_{ff} sizes even at greater depths. This supports prior theoretical work Kim et al. (2023); Vardi & Shamir (2022) suggesting that multi-head attention improves the capacity to memorize arbitrary sequences, especially when models are restricted by the total number of parameters.

Furthermore, transformers achieve memorization thresholds comparable to or exceeding MLPs of similar size (such as a threshold of 77 at 64k params with 32 heads versus 1728 for MLPs at 30k params). However, they achieve this with smaller d_ff sizes, leveraging the power of attention rather than pure width.

In summary, transformers perform poorly compared to MLPs on memorization tasks when total parameter count is fixed, suggesting that attention may not be an efficient use of parameters in this task. While attention helps with generalization and learning long-range dependencies, it seems to hurt performance when the goal is just to memorize random mappings. MLPs, despite being

simpler, allocate parameters more similarly to storing information, giving them an advantage under a constrainted parameter count.

Table 4: Transformer Results (32,000 parameters)

hidden_size	d_ff	num_heads	num_layers	Mean Threshold	Median Threshold	Std. Dev.
32	380	1	1	11.8	12	0.45
32	380	2	1	16.4	16	0.89
32	380	4	1	18	18	0
32	380	8	1	17	17	0
32	128	1	2	12.4	12	1.82
32	128	2	2	16.4	16	0.55
32	128	4	2	17.6	17	2.51
32	128	8	2	17	17	0.71
32	40	1	3	9.2	11	4.02
32	40	2	3	12.4	12	2.88
32	40	4	3	16.8	17	0.45
32	40	8	3	17.2	17	1.79

Table 5: Transformer Results (64,000 parameters)

hidden_size	d_ff	num_heads	num_layers	Mean Threshold	Median Threshold	Std. Dev.
64	768	1	1	18	16	6.00
64	768	2	1	22.4	20	3.29
64	768	4	1	57.4	63	7.67
64	768	8	1	37.2	31	11.28
64	768	16	1	31	31	0
64	768	32	1	77	77	0
64	256	1	2	19.4	17	7.57
64	256	2	2	28.6	27	2.19
64	256	4	2	29	24	11.18
64	256	8	2	32	32	0
64	256	16	2	64	64	0
64	256	32	2	57.4	64	14.76
64	83	1	3	6	4	4
64	83	2	3	24.2	24	1.10
64	83	4	3	24.8	26	2.68
64	83	8	3	27	27	6.04
64	83	16	3	48	48	11.42
64	83	32	3	41	41	0

Table 6: Transformer Results (128,000 parameters)

hidden_size	d_ff	num_heads	num_layers	Mean Threshold	Median Threshold	Std. Dev.
128	1550	1	1	22.8	24	1.79
128	1550	2	1	24.6	23	3.58
128	1550	4	1	49	49	0
128	1550	8	1	22	22	0
128	1550	16	1	32	32	0
128	1550	32	1	31.2	32	1.10
128	1550	64	1	31	31	0
128	512	1	2	16.6	16	3.29
128	512	2	2	16.6	20	7.60
128	512	4	2	20	20	0
128	512	8	2	23	21	4.47
128	512	16	2	30	30	0
128	512	32	2	32	32	0
128	512	64	2	42.4	42	13.07
128	175	1	3	16	16	0
128	175	2	3	20	20	0
128	175	4	3	18.8	17	2.49
128	175	8	3	26.6	25	4.10
128	175	16	3	24	23	1.73
128	175	32	3	30.6	23	18.12
128	175	64	3	31	31	0

7 Conclusion and Future Work

We have discussed the following key takeaways for MLPs and transformers.

For MLPs, width has a key role in memorization capabilities. At a fixed parameter count, the widest and shallowest models consistently perform the best. In particular, MLPs with a depth of 1 outperform deeper configurations. As depth increases and width becomes more narrow, optimization becomes more difficult, leading to higher variance and sudden drops in memorization accuracy. These findings align with theoretical scaling laws and emphasize the idea that memorization is sensitive to width constraints.

Transformers, however, show a much different behavior. Adding more attention heads at a fixed depth significantly improves memorization, even when total parameter count is constrained. Interestingly, the performance of transformers does not degrade as quickly as MLPs when decreasing width and increasing depth, suggesting that a narrow width is less harmful when paired with attention mechanisms. Moreover, the benefits of increasing the number of attention heads are nonlinear: performance tends to plateau or even worsen beyond certain thresholds, such as with 32 heads or a total parameter count of 64k. Overall, while attention contributes to better memorization in specific configurations, transformers remain less effective than MLPs in pure memorization tasks, especially when dealing with random dictionary mappings. This suggests that attention mechanisms may be unnecessary in pure memorization tasks with random dictionary mappings.

For future work, we plan to explore a more traditional transformer configuration by holding the feedforward dimension d_{ff} fixed at four times the hidden dimension ($d_{ff} = 4 * d\_model$). This design is common in other literature, and it may help to examine how scaling the hidden size affects understanding memorization and parameter trade-offs.

Author Contributions

All authors contributed equally to the conceptualization and experimental design of the memorization capacity study. Diya Hundiwala implemented the MLP experiments, conducted data analysis for width and depth trade-offs, and prepared the results tables. Christine Guo implemented the code, transformer experiments, focusing on the role of attention heads and depth, and prepared visualizations and plots. Niranjan Vijaya Krishnan assisted with debugging, hyperparameter tuning, and

supported literature review efforts. All authors participated in interpreting results, writing the report, and revising the manuscript collaboratively. Diya Hundiwala wrote the MLP Results and Related Works sections, Christine Guo wrote the Methodology and the Transformer Results sections, and Niranjan Vijaya Krishnan also wrote the Related Works and Parameter Selection sections.

REFERENCES

Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, A Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.

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

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. *arXiv preprint arXiv:2012.07805*, 2021.

Nicholas Carlini, Florian Tramer, Matthew Jagielski, Ariel Herbert-Voss, Tom Brown, Ulfar Erlingsson, Dawn Song, et al. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Vitaly Feldman. Does learning require memorization? a short tale about a long tail. *arXiv preprint arXiv:1906.05271*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Tokio Kajitsuka and Issei Sato. On the optimal memorization capacity of transformers. In *International Conference on Learning Representations*, 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Seungjae Kim, Hoyeon Lee, Joonseok Park, and Jinwoo Shin. Capacity of transformers: A theoretical and empirical study. *arXiv preprint arXiv:2305.02314*, 2023.

Xingyu Lu, Xiaonan Li, Qinyuan Cheng, Kai Ding, Xuanjing Huang, and Xipeng Qiu. Scaling laws for fact memorization of large language models. *arXiv preprint arXiv:2406.15720*, 2024.

Sadegh Mahdavi, Renjie Liao, and Christos Thrampoulidis. Memorization capacity of multi-head attention in transformers, 2024. URL https://arxiv.org/abs/2306.02010.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2020.

Hyunghoon Park, Tianlin Zhao, and Venkatesh Saligrama. The power of depth for feedforward neural networks. *arXiv preprint arXiv:2106.03348*, 2021.

Jackson Petty, Sjoerd van Steenkiste, Ishita Dasgupta, Fei Sha, Dan Garrette, and Tal Linzen. The impact of depth on compositional generalization in transformer language models, 2024. URL https://arxiv.org/abs/2310.19956.

Alec Power, Yuri Burda, Harri Edwards, Owain Evans, and Nicholas Joseph. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Alon Vardi and Ohad Shamir. Optimal memorization in neural networks: The power of depth and precision. *arXiv preprint arXiv:2202.08434*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2017.

A Appendix

Please see this link for access to the code used to run the MLP and transformer experiments (download this as a pdf and click on the word link).