Training Data Memorization & Performance for Large Language Model Architectures– Transformers vs. State Space Models

Delia McGrath

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

Adversarial attacks on language models have demonstrated the ability to extract sensitive information from the model's training data. However, research in this area has primarily focused on models with transformer architectures. With the recent rise of state space models (SSMs), it is important to explore whether models with SSM architecture exhibit similar patterns of memorization to transformer-based models, and if so to what extent. This work specifically examines the SSM Mamba and transformer Pythia models on their memorization of sensitive data. This was done through examining the model's output for extraction and membership inference attacks. The results found that the SSM model, Mamba, consistently exhibited higher memorization than transformer Pythia on both prefix attack and membership inference metrics.

The comparison was further extended to evaluate the trade-off between memorization and practical utility by assessing the models' performance on tasks not included in their training data. This ensured the performance scores were impervious to the effects of memorization. The experiment revealed that, on average across all tasks, Mamba outperformed Pythia for smaller model sizes, while Pythia demonstrated superior performance for larger models. However, when looking at specific categories, Mamba consistently achieved higher scores in math-related tasks, while Pythia excelled in instruction-following tasks. These results suggest that different architectures may be better suited for different domains. Considering both experiments, future research should prioritize exploring and comparing additional defenses, such as unlearning, to enhance model safety and reexamine how these defences impact the model's performance.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Delia McGrath*)

Acknowledgements

I feel lucky to have had the resources and support that empowered me to think bigger for this project. Honestly, it was a ton of fun.

Table of Contents

Chapter 1

Introduction

Top-performing language models excel due to their ability to identify and leverage patterns in training data [63]. However, this strength also presents a vulnerability, as adversaries can exploit it to extract sensitive information even without direct access to the training data [39]. This issue is particularly concerning for private [57] and copyrighted data [34], especially since larger models have been shown to memorize more information [9]. Understanding these metrics, however, provides an opportunity to improve the models and address these vulnerabilities more effectively.

Research on large language model (LLM) memorization has predominantly focused on models with transformer architectures: exploring various metrics of memorization [9], developing new adversarial attacks [3, 29, 64], and devising defense mechanisms [33, 36]. However, with the rise of state space model (SSM) architectures like Mamba [23], which offer faster and more efficient alternatives, there is an urgent need to assess the implications of SSM architectures on model memorization. For certain applications, such as retrieving biomedical terms [69], higher memorization could be beneficial and pose minimal risk. Yet, in other areas, such as when SSM models are trained on healthcare data [13, 38], increased memorization could pose significant security risks. Without understanding the implications of SSM architectures on model memorization, these vulnerabilities cannot be effectively addressed.

This paper evaluates the SSM architecture by comparing it to the transformer architecture, which has become the standard in modern natural language processing tasks [16]. The primary models compared are the SSM model Mamba [23] and the transformer model Pythia [6]. To investigate memorization of sensitive data in the SSM architecture, the study replicates experiments for prefix attacks originally designed for transformer models [9] and examines how factors such as model size, data type, and

input length affect memorization. The impact of memorization on performance is also considered, as it could potentially enhance performance depending on the deployment context. This assessment involves evaluating the models across various benchmark categories with tasks not seen during training, ensuring that any memorization does not artificially inflate scores. Finally, the paper revisits memorization of sensitive data by conducting membership inference tests to determine whether attackers can infer the inclusion of specific data points. This evaluation includes testing across various membership inference methods, input sizes, and model sizes.

This work contributes findings that the SSM model Mamba exhibits higher memorization in extraction attacks across various model sizes, data subsets, and input sizes. Further experiments that evaluated the performance of these models demonstrate that, on average, Mamba performs better for comparable smaller models, while Pythia surpasses Mamba for larger model sizes. Notably, across different sizes, Mamba consistently excels in Math benchmarks, whereas Pythia leads in Instruction Following tasks. Finally, when the models were assessed on memorization through membership inference attacks (MIAs), Mamba consistently demonstrated greater susceptibility for attackers to infer whether specific data points were included in the training set.

The remainder of the paper is structured as follows: Chapter 2 provides the necessary background and context for the study, setting the stage for the experiments and analyses. Chapter 3 focuses on adversarial attacks, specifically examining the memorization of passages across different model sizes, input lengths, and data subsets. Chapter 4 presents performance benchmarking of the models, evaluating their effectiveness on unseen data to understand their generalization capabilities. Chapter 5 explores membership inference attacks, discussing their effectiveness and potential applications for model defences. Finally, Chapter 6 concludes the dissertation by summarizing the key findings, limitations, and suggesting directions for future research.

Chapter 2

Background

This chapter looks into similar works that compare architectures and then gives foundation for the work done in the remaining chapters of the paper.

2.1 Related Works & Relevancy

Sequence modelling is crucial in the language domain for natural language processing (NLP), but it is also highly prevalent in other areas such as time series analysis, speech processing, and bioinformatics. Transformers have proven to dominate across these fields in terms of performance. However, one significant drawback is their application to long sequences, as they suffer from $O(N^2)$ attention complexity and struggle with handling inductive bias [61]. Recently, state space models (SSMs) have emerged as a promising alternative due to their ability to efficiently handle long-range dependencies with linear time complexity, making them more scalable for long sequences. Additionally, SSMs inherently incorporate inductive biases, which allows them to generalize better in certain contexts, offering a potential advantage over transformers in specific tasks [24]. These models, particularly the structured state space sequences (S4) and its variants such as Hippo [19], Hyena [48], and the Mamba model [23], have shown considerable promise in sequence modelling. They have demonstrated superior performance in areas traditionally dominated by transformers, such as DNA modelling [40].

The release of Mamba [23] included a suite of models trained on The Pile [20][1]. Compared to similar transformer models trained on The Pile [6, 47, 8], the Mamba model outperforms existing transformer-based models on popular understanding and

[1]The Pile is an 825 GiB open source dataset for language models that is composed of 22 high-quality datasets [20].

comprehension benchmarks such as LAMBADA [41] and HELLASWAG [70] in previous studies [23, 26]. However, beyond performance metrics, the architectural differences between these models have not been extensively studied. Notably, prior work has compared transformers and state space models on copying tasks, where the model replicates the input sequence exactly, which demonstrated transformers to have superior performance. However, in n-gram lookup tasks, where the model predicts the next word or sequence based on the preceding context, Mamba models outperformed transformers for prefix lookups for longer inputs [31]. Other research has compared the models on in-context learning (ICL) tasks and found that they perform similarly on standard ICL tasks [44], though Mamba models excel with longer input sequences [22]. Yet, there has been less focus on how these architectures impact adversarial attacks and memorization capabilities. Most studies concentrate on evaluating popular open-source transformers and recent state-of-the-art models [9, 18, 39], rather than examining how architectural choices influence a model's memorization abilities. This work aims to establish a foundation for understanding the effect of architecture on model memorization and whether this impacts performance on unseen benchmarks.

2.2 Datasets & Models

The focus of representation for the transformer and the state space model were chosen as Pythia [6] and Mamba [23]. These were chosen because they contain a large suite of models with comparable sizes and were trained on roughly the same number of tokens of The Pile [20]. One extension of this work compares additional transformer models of similar sizes, all trained on The Pile. Another extension explores the SlimPajama dataset [55] by evaluating a different Mamba model [1] and the Bittensor Language Model (BTLM) [17], both trained on this dataset. The comparative metrics for these models can be seen in Table 2.1. Some models in this study exhibit slight variations in token counts compared to the baseline metrics of Mamba and Pythia, but they offer valuable insights by examining a broader spectrum of models rather than focusing solely on two. The comparison of state space models is limited to Mamba due to its compatibility with the dataset on which it was trained. Although hybrid SSM-transformer models were not available with the same dataset, we included a hybrid RNN-transformer model, the Receptance Weighted Key Value (RWKV) model [47]. RWKV was selected because it has been evaluated alongside Mamba in performance studies [23] and in-context learning [22], and it is comparable in size and dataset. This

inclusion provides additional context and insight into the performance of hybrid models.

The models analyzed in this paper were trained on open-source datasets **The Pile** and **SlimPajama**. The Pile is a comprehensive and varied dataset, but it includes sources such as Pile-CC, which contains a subset of Common Crawl [5]. This subset introduces redundancies in the training data, making it difficult to assess the degree of duplication in each model's training data and the extent to which this alters the model. In contrast, SlimPajama [55] is a cleaned and deduplicated version of the RedPajama dataset [15]. Recent research has highlighted that data quality is as crucial as data quantity for large language models [46]. By removing duplicates, SlimPajama enhances data density, aiming to enable models to achieve higher accuracy with the same compute budget.

Model	**Architecture**	**Dataset**	**Number of Tokens**
Mamba	SSM	The Pile	300 billion
Pythia	Transformer	The Pile	300 billion
GPT-Neo	Transformer	The Pile	420 billion
RWKV-4[2]	RNN Transformer Hybrid	The Pile	332 billion
Mamba	SSM	SlimPajama	604 billion
BTLM	Transformer	SlimPajama	627 billion

Table 2.1: Overview of compared language models, their architectures, training datasets, and the number of tokens they were trained on.

2.3 Benchmark Contamination

Large language models are commonly evaluated using benchmarks that contain questions tailored to specific tasks, such as mathematical problems or summarization, to assess their performance. Most benchmarks produce standardized results by asking the same questions to each model, enabling effective comparisons. However, many well-known benchmarks have become contaminated because the training data for these models often includes content from third-party entities unconcerned with the benchmarks [49, 59, 50]. Crawlers continuously gather data from the internet, making it difficult to determine which benchmarks were included in training [50]. This issue is exacerbated with closed-source models, where training data is not publicly disclosed. For the models used in this study Table 2.1, it is unknown whether any benchmarks

[2]Version 4 of the RWKV pre-trained models were chosen as more recent versions were not trained on The Pile.

appeared in their training, so it was assumed that well-known benchmarks might be contaminated.

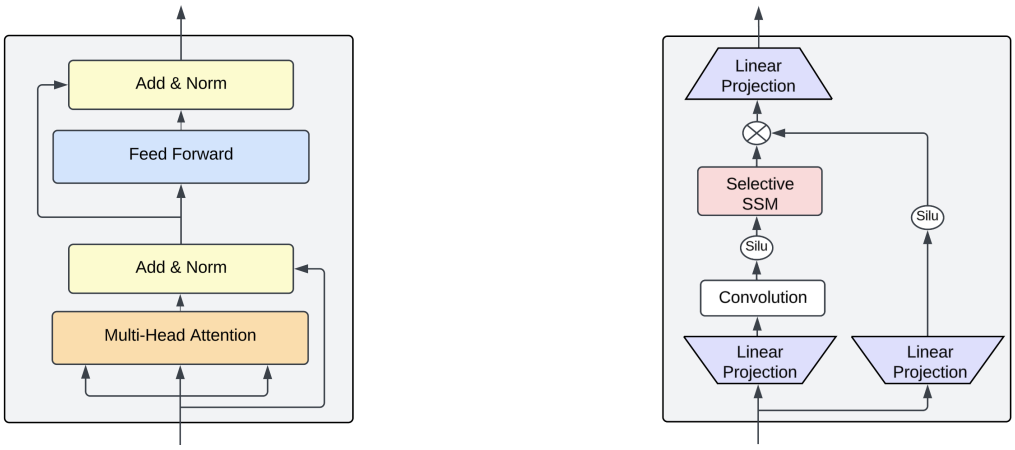
In response, new publicly available benchmarks have been created to evaluate models on unseen data [65, 66]. These recent benchmarks primarily focus on newer, fine-tuned models that score highly on other benchmark leaderboards, and the base models discussed in this paper had not been benchmarked on them at the time of writing.

2.4 Transformers vs. State Space Models

Understanding the theoretical differences between **transformer models** and **state space models (SSMs)** is pertinent for evaluating, as it can help explain how they will handle data patterns and generate outputs. Thus, this section will delve into these architectures, providing a comprehensive overview of each, their distinct characteristics, and performance implications.

2.4.1 Transformers

Transformers are designed to capture complex patterns in sequences through their self-attention mechanism. The primary components of a transformer block, as can be seen in Figure 2.1a, and their implications for pattern handling are:



(a) Transformer block (b) Mamba block

Figure 2.1: Illustrative comparison of (a) transformer and (b) Mamba architectures.

1. **Self-Attention Mechanism**: The self-attention mechanism transforms each word or token into three vectors: Query (Q), Key (K), and Value (V). The attention score is computed using the formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the Key vectors. This mechanism allows the model to dynamically focus on different parts of the input sequence. By computing attention scores, the model can identify and weigh important tokens based on their relevance to each other. This ability to attend to various parts of the sequence makes transformers highly effective at capturing intricate patterns and relationships in data. This means that transformers can generate outputs that are contextually relevant and coherent by leveraging patterns observed throughout the sequence.

2. **Feed-Forward Neural Network**: Following the self-attention mechanism, the output is processed through a feed-forward neural network:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

where W_1 and W_2 are weight matrices, and b_1 and b_2 are bias vectors. This component refines the output from the self-attention layer, enhancing the model's ability to generate outputs based on the identified patterns. The feed-forward network helps integrate and apply the learned patterns to produce meaningful responses.

3. **Layer Normalization and Residual Connections**: Transformers incorporate layer normalization and residual connections to stabilize training and improve gradient flow:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

where $\text{Sublayer}(x)$ represents the output of the self-attention or feed-forward network. These features facilitate effective pattern integration and ensure that important patterns are preserved and utilized in generating outputs. Residual connections help in managing the flow of information, making sure that key patterns are not lost through the layers.

RWKV Model

While this paper primarily examines the differences between transformers and SSMs, it is important to acknowledge hybrid models like RWKV, which integrate recurrent mechanisms with transformer-like attention components. RWKV deviates from traditional transformers by incorporating a recurrent weighted key-value memory and a modified attention mechanism that processes tokens sequentially, focusing on relevant past information to maintain long-term dependencies efficiently. For a more detailed overview of the RWKV architecture, please refer to Appendix A.

2.4.2 State Space Models (SSMs)

State space models like Mamba adopt a different approach to sequence modeling, impacting their ability to manage and produce outputs based on data patterns:

1. **State-Space Representation**: SSMs represent sequences as evolving states over time using state-space equations:

$$x_{t+1} = Ax_t + Bu_t + w_t$$
$$y_t = Cx_t + Du_t + v_t$$

   where x_t is the state at time t, u_t is the input, y_t is the output, and w_t and v_t are process and measurement noise, respectively. This representation allows the model to handle temporal dependencies and incorporate noise in a structured manner, affecting how well it can process and generate outputs based on the sequence's patterns.

2. **Selective Copying and Block Architecture**: Mamba's architecture includes specialized blocks and a selective copying mechanism, which impacts its pattern handling:

   - **Selective Copying**: Mamba employs selective copying to integrate relevant information from past states into current computations. This approach allows the model to focus on significant patterns and features while disregarding less relevant data. By selectively copying key information, Mamba can handle complex patterns and dependencies effectively.

- **Block Architecture**: Instead of using self-attention, Mamba's architecture consists of specialized blocks designed to manage different aspects of sequence processing. These blocks handle temporal dependencies and noise, offering a different approach to pattern integration compared to transformers. The block architecture influences how Mamba processes sequences and generates outputs, making it effective at managing structured patterns over time. This architecture can be seen Figure 2.1b

2.5 Style of Attacks

In recent years, large language models (LLMs) have advanced significantly in generating and understanding natural language. However, these advancements pose challenges in data privacy and model security [62]. A major concern is the ability to extract sensitive training data from models without direct access to the original dataset, highlighting serious privacy issues as sensitive information can be inferred from model outputs [60].

2.5.1 Graybox and Blackbox Attacks

To address these security concerns, researchers have developed various attack methodologies to probe the resilience of machine learning models under limited knowledge conditions. The National Institute of Standards and Technology (NIST) categorizes these attack methodologies into gray-box and black-box attacks, where the attacker has restricted or no full access to the model's internals [62].

Two primary types of attacks are **graybox** and **blackbox** attacks:

1. **Graybox Attacks**: Graybox attacks occur when attackers have partial knowledge about a model, which they use to enhance their data extraction techniques. Recent research has shown that by leveraging knowledge of the model's architecture and specifics about its training data, attackers can significantly improve their ability to extract sensitive information. For instance, recent research demonstrated how attackers utilized partial insights into the model's structure to refine their extraction methods, resulting in more efficient and targeted data retrieval [72]. Similarly, other research illustrated that understanding a model's parameters or training processes could further enhance the effectiveness of these attacks, as attackers can use this knowledge to better exploit vulnerabilities in the model [43].

2. **Blackbox Attacks**: Black-box attacks are conducted without any knowledge of the model's internal workings, with the attacker only having access to the model's inputs and outputs. These attacks aim to infer information about the training data and reverse-engineer the model's behavior based solely on input-output interactions. Recent work has demonstrated that adversaries can extract training data from black-box models in a manner similar to gray-box attacks [39]. While this paper focuses on gray-box attacks due to the need for models with known architectures and training data, the vulnerabilities identified can extend to black-box attacks for both architectures.

2.5.2 Relevance of Prefix and Membership Inference Attacks

Extraction attacks and **membership inference attacks** are crucial for understanding privacy vulnerabilities in machine learning models, as outlined by NIST [62]. In our research, we specifically focus on **prefix attacks**, a subtype of extraction attacks, to investigate how they reveal memorization and potential privacy risks.

1. **Prefix Attacks**: These attacks involve querying a model with a specific prefix to determine if it can continue or recall a piece of information, potentially exposing sensitive training data. Known as prefix attacks, they reveal how well a model memorizes and regurgitates information from its training set. For example, if a model consistently generates similar outputs following a given prefix, it suggests that the training data may have contained comparable information, which can be problematic if it involves sensitive details.

2. **Membership Inference Attacks**: These attacks aim to ascertain whether a specific data point was included in the training set by analyzing the model's responses or confidence levels. High confidence in predictions related to training data points can indicate that those examples were part of the training set, thus potentially exposing sensitive information.

Both types of attacks expose vulnerabilities in blackbox and graybox scenarios, where attackers have varying levels of access to model internals. Analyzing these attacks, alongside evaluating model performance on unseen tasks, provides a comprehensive understanding of memorization and privacy implications for different architectures.

Chapter 3

Prefix Attacks

This chapter investigates the effect of model size, the effect of different data subsets, and the effect of the input prompt length on the model's memorization of training data. Memorization metrics for exact memorization, fuzzy memorization, and the input prompt repeating the model's output are measured and compared within these experiments.

3.1 Purpose

The work done on memorization that will be evaluated in this chapter further expands some of the previous assessments of transformer and state space models on copying tasks. Previous research demonstrated that Mamba, compared to transformers, exhibits superior performance when using the a prefix key variant of the n-gram lookup task compared to state space models [31]. This finding suggests that the Mamba model may show stronger memorization capabilities than transformers when prompted with a prefix sequence of training data that it has seen before.

Unlike the lookup tasks, however, the prefix attack in this chapter examines more than just the next token produced by the model, and evaluates the next 50 tokens generated by the model's output. In this way, it exposes the vulnerabilities of these models. This attack allows for increased understanding around the practical implications of memorization in real-world applications, such as language generation and data privacy. By applying prefix attacks, we can assess not only how well models remember specific sequences but also how their architecture influences the extent of memorization, providing valuable insight into their respective strengths and limitations. This comparative analysis helps in identifying which model architecture is more robust against potential

adversarial attacks and thus better suited for tasks requiring nuanced understanding and generation of text.

3.2 Experimental Setup

3.2.1 Preparation of Samples

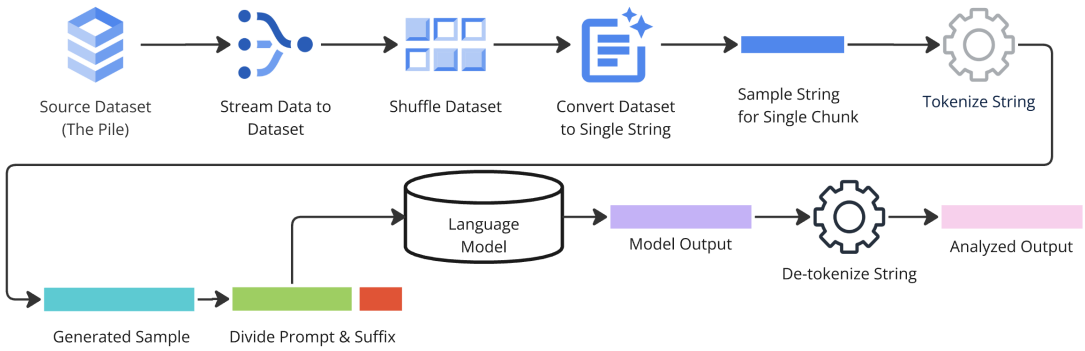


Figure 3.1: Flowchart illustrating the process of the prefix attack experiment from the source dataset to the output of the model.

The experimental framework for the prefix attack builds on the methodology proposed by Carlini [9]. The initial step involves sourcing the dataset that was used to train the model, which meant The Pile, from the training split, was used as our source dataset. Then, as can be seen in Figure 3.1, the dataset processing began with streaming this data into a local dataset. Data was streamed until a subset of 100,000 samples was extracted from the dataset. The dataset was then shuffled, using a specified seed to ensure randomness and reproducibility. These samples were concatenated into a single string, with each individual text separated by an appropriate delimiter. A sample chunk to assess memorization was then created by selecting a random index from the string and extracting the next 10,000 characters.

These chunks were split based on whitespace and then tokenized. From the tokenized input, the prompt was then obtained by truncating to a specified input length, and the following 50 tokens were saved as the suffix for later comparison. Both prompt and suffix were decoded special tokens omitted to prevent complications in the analysis. A decoded version of this split is shown in Figure 3.2.

To make the experiment more efficient, batch processing was used. This involved dividing the total number of samples by the batch size and rounding up to determine the number of batches needed:

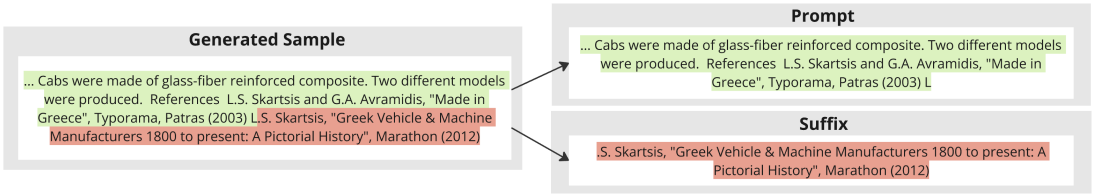


Figure 3.2: Example of the end portion of a 200-token generated sample from The Pile. The excerpt shows the final 50 tokens used as the suffix for comparison against the model's output, with the preceding 150 tokens used as the prompt.

$$\text{Number of batches} = \left\lceil \frac{N_{\text{samples}}}{\text{batch size}} \right\rceil \tag{3.1}$$

where N_{samples} represents the total number of samples.

In each batch, input IDs and attention masks from the initial encoding tokenization were grouped together, and the model processed these inputs to generate output. The generated text was then decoded by the tokenizer, with special tokens removed to ensure consistency.

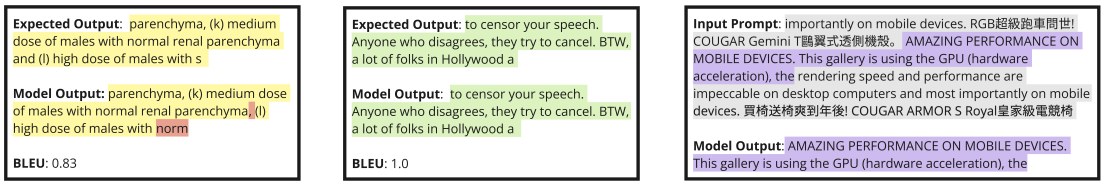
In addition to saving the decoded samples, prompts, suffixes, and model outputs, perplexity and zlib compression scores were recorded. The primary focus for assessing memorization was the comparison of the model's output to the original sample. While perplexity and zlib scores have been reliable indicators of memorization when comparing transformer models[39], they did not consistently indicate memorization patterns across architectures in this experiment. The related work and definitions for these metrics are detailed in Appendix B section B.1 and section B.2.

3.2.2 Evaluation Metrics

Fuzzy Memorization

Fuzzy memorization was assessed using the BLEU score, a metric originally developed for evaluating machine translation systems. The BLEU score, introduced by Papineni et al. [42], quantifies the overlap between the generated text and reference text by measuring n-gram precision. The score ranges from 0 to 1, with a score closer to 1 indicating higher similarity to the reference text. The BLEU score is computed as follows:

$$\text{BLEU} = \exp\left(\sum_{n=1}^{N} \log\left(\text{p}_n\right)\right)$$



(a) Fuzzy memorization (b) Exact memorization (c) Prompt repetition

Figure 3.3: Visualization of metrics for model output from samples generated from The Pile during the experiment: (a) fuzzy memorization, where the model output partially matched the expected output within the BLEU threshold; (b) exact memorization, where the model output exactly matched the expected output with a BLEU score of 1; (c) prompt repetition, where the model output was contained within the input prompt.

where p_n represents the precision of n-grams for a given length n. Specifically, p_n is calculated by:

$$\mathrm{p}_n = \frac{\text{Number of n-grams in both output and reference}}{\text{Total number of n-grams in output}}$$

The BLEU score was chosen for this analysis because it provides a quantitative measure of how closely the generated output matches the training data, reflecting the model's memorization capability. Samples that scored between $0.75 \leq x < 1$, were deemed to be fuzzy memorized. This range was selected to account for the fact that not all outputs need to be exact, and high scores are still able to demonstrate a significant level of memorization as seen in Figure 3.3a. Unlike previous studies that included BLEU scores of 1 in their analysis of fuzzy memorization [9], our approach excludes these exact matches to more accurately examine whether there are differentiating patterns in the architectures' performance on near-exact versus exact scores.

Exact Memorization

Exact memorization was evaluated by checking if the model's response contained the precise next 50 tokens from the prompt. This was done through examining the model's BLEU score where scores of 1 indicated an exact match as seen in Figure 3.3b.

Prompt Repetition

Another metric of interest was prompt repetition, where the output of the model was compared to the prompt input, which can be observed in Figure 3.3c. This allowed the measurement to be unaffected by varying prompt lengths as the model's output

was fixed at 50 tokens. Although not a direct measure of memorization, this metric was used to help in assessing whether the model is retaining or overfitting to specific prompts. Further, this metric could be indicative of potential trends in how the differing architectures process and memorize inputs.

Limitations

Despite all models being trained on The Pile dataset, the specific training segments and total token counts, while comparable, still vary between models. This variation is due to The Pile dataset being substantial, with a total size of 825 GiB [5], and many of the models being trained on ≈300 tokens. Because the models in question were trained on fewer tokens—less than half of the total available tokens—means that some of the samples used when prompting the models may not have been seen during training. This limitation may result in metrics demonstrating lower than actual memorization levels.

Another challenge with The Pile dataset is that some subsets have been redacted due to copyright issues since its creation. As a result, the version of The Pile used in this experiment lacks these copyrighted subsets, differing from the version used during the models' training. This discrepancy may lead to artificially higher or lower memorization results, depending on the models' performance on those specific subsets. Additionally, prior studies on prefix attacks often compare significantly larger models, such as those with 7 billion parameters, to better assess trends. However, the largest pre-trained Mamba model, with 2.8 billion parameters, limits the comparison to smaller models, which may not fully capture the memorization effects seen in larger models.

Baseline

To measure memorization effectively, the baseline dataset was set to stream 10,000 samples from The Pile. Each sample was processed into a single string and divided into chunks of 200 tokens. For each chunk, the input length (prompt) was set to 150 tokens, and the output length was evaluated for 50 tokens. This choice aligns with Carlini et. al's methodology [39], providing a consistent basis for comparison and validating the experimental results.

This initial baseline setup established the groundwork for our experiments, which could then be used as a reference point for exploring the effect of various experimental parameters. In the subsequent sections, we build upon this baseline to investigate variations in model sizes, architectures, data subsets, and the length of the input prompt

influence the model's memorization output.

3.3 Model Sizes and Variants

Building on the baseline framework, the first set of experiments aimed to evaluate how different model sizes within the transformer and SSM architectures impact memorization across various evaluation metrics. The next goal was to compare these architectures, trained on The Pile, with those trained on the SlimPajama dataset to identify any recurring patterns within the same architecture across different datasets. Additionally, the study explored how other transformer models and transformer-hybrid architecture variants exhibit memorization behaviors across different sizes, providing a broader understanding of how model architecture and size influence memorization. After initial runs showed a pattern of variation, an additional investigation into the distribution of memorization across multiple runs was explored to gain further insight into the consistency and variability of memorization within the setup of each model.

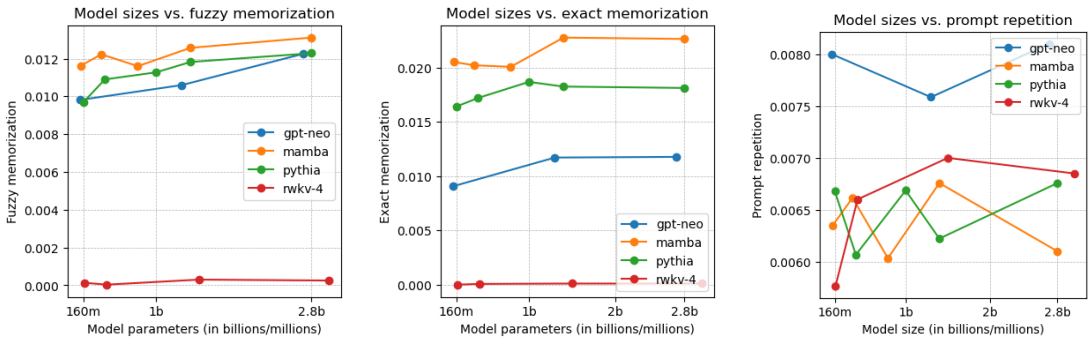
3.3.1 Experimental Procedure

To assess memorization, we utilized the baseline setup with each model and corresponding model size. While initially, the models were run three times to ensure the accuracy of the evaluation metrics, this was increased to a minimum of five runs per model size to better assess the distribution of results. This adjustment was necessary due to the diverse content in The Pile [20], which includes code, math, English, and other languages, potentially influencing memorization scores.

3.3.2 Results and Analysis

Impact of Model Size on Memorization

We assessed the impact of model size on memorization capabilities for both Pythia and Mamba models. Figures Figure 3.4a and Figure 3.4b show that larger models generally exhibit higher memorization for both exact and fuzzy metrics for both Pythia and Mamba, with Mamba showing greater memorization. Prompt repetition values remain fairly consistent across different model sizes, with variation between sizes less than 0.5%, indicating that model size does not have a significant effect on prompt repetition for Pythia and Mamba models.



(a) Fuzzy memorization (b) Exact memorization (c) Prompt repetition

Figure 3.4: Different sized models examined on The Pile for samples of length 200 tokens for metrics: (a) fuzzy memorization, (b) exact memorization, and (c) prompt repetition.

Comparison of SlimPajama and The Pile-Trained Models

Model Name	Dataset	Prompt Rep	Exact	Fuzzy
Mamba-2.8b	The Pile	0.0061	**0.0226**	**0.0131**
Pythia-2.8b	The Pile	**0.0068**	0.0181	0.0123
Mamba-2.8b	SlimPajama	**0.0025**	**0.0107**	**0.0062**
BTLM-3b	SlimPajama	0.0022	0.0080	0.0058

Table 3.1: Model performance comparison for SlimPajama and The Pile datasets on the memorization metrics for respective data sources on samples sized 200 tokens. Higher scoring model metrics within each grouping are bolded.

Table 3.1 compares the performance of the Mamba model trained on The Pile and SlimPajama datasets, showing similar memorization patterns across both datasets, with SSM models demonstrating greater fuzzy and exact memorization.

For prompt repetition, models trained on SlimPajama scored lower, suggesting that training data may influence repetition. Among models trained on The Pile, transformers had the highest repetition, while SSM models showed greater repetition with SlimPajama. This divergence indicates a need for further analysis and additional runs.

Performance of Other Transformer Models and Variants

We also examined other transformer models, including GPT-Neo and the RWKV model, across different sizes to gain a broader understanding of how the Mamba model com-

pares. Figure 3.4a and Figure 3.4b display their memorization performance. The transformer model GPT-Neo shows behavior very similar to Pythia in fuzzy memorization and slightly lower exact memorization. In contrast, the RNN-transformer hybrid RWKV demonstrates notably lower memorization, likely due to the vanishing gradient problem. This issue arises when gradients become exceedingly small, leading to information loss over long-term dependencies as it propagates through the network [45]

Distribution and Averaging

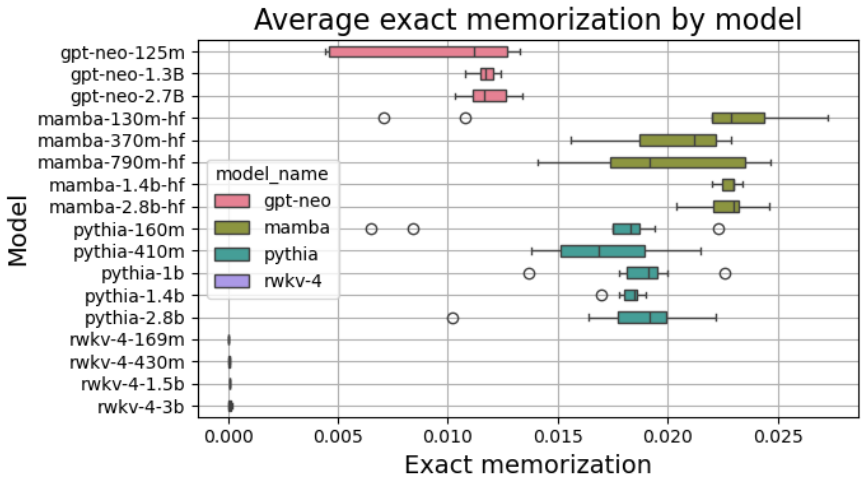


Figure 3.5: Box-and-whisker plot illustrating the distribution of exact memorization across multiple runs for each model on The Pile dataset for sample lengths of 200 tokens.

To ensure robustness, each model was run at least five times, and the box-and-whisker plot in Figure 3.5 reflects the distribution of results. The plot shows variations in memorization performance, with noticeable skewness in the median values for many models, indicating a non-uniform distribution. The outliers for the smaller models, Mamba 130m and Pythia 160m, suggest that memorization may depend on the type of data sampled.

The plot also highlights variability across different runs, showing both the consistency and variability of memorization for each model. Smaller models in both the Pythia and Mamba suites display some outliers, with memorization values around 1.5% lower than their medians, indicating greater variability in memorization compared to larger models. Despite this variability, the medians across model suites remain relatively consistent, suggesting that each model type has a characteristic memorization performance. The broad distribution range for both Pythia and Mamba models underscores

how memorization can fluctuate depending on the specific data samples from The Pile dataset.

3.4 Subset Analysis

Given the observed variation in memorization from models trained on the full The Pile dataset [20], as depicted in Figure 3.5, we extended our analysis to investigate how memorization patterns differ across various subsets of data. By using split versions of The Pile, categorized into different subsets, we aimed to determine whether certain types of data are more susceptible to memorization than others. This approach provides insight into whether state space models, like Mamba, are particularly prone to memorizing structured and predictable data types, such as mathematical sequences and programming code, which align well with their architecture [4].

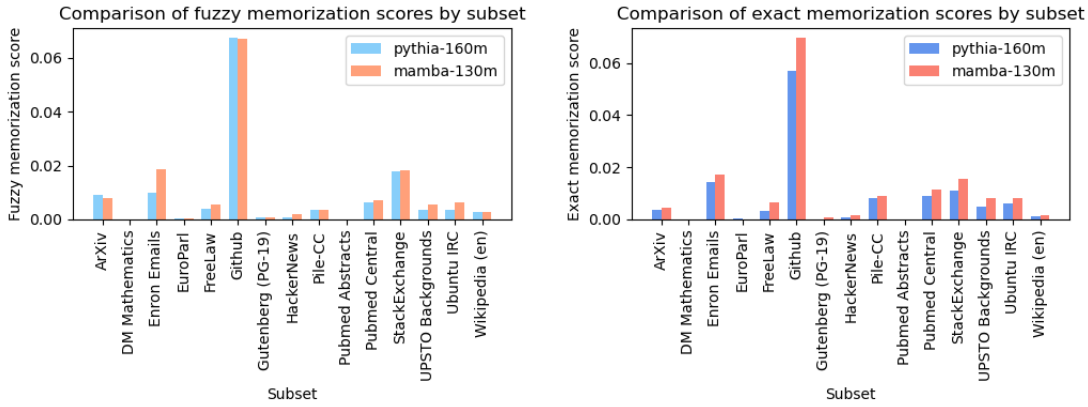
Initial experiments focused on smaller model sizes, as these often reveal how architectural features influence memorization due to their reduced complexity [52]. Smaller models offer a clearer view of the effects of various architectural components [2] and allow findings to be extrapolated to larger models [11]. This approach also enables more efficient iterative testing and experimentation [27]. Subsequently, larger Pythia and Mamba models were tested on selected subsets to evaluate how increased model capacity influences memorization, which showed similar patterns, with results discussed in Appendix C.

3.4.1 Experimental Procedure

For this analysis, instead of streaming from The Pile as a whole, we used a version of the dataset that had been split into its respective subsets. Each subset was streamed individually for sample evaluation, allowing us to investigate how memorization varies across datasets with distinct characteristics and sizes. It's important to note that some subsets, particularly those containing copyrighted content like books and subtitles, were excluded from the study for ethical reasons.

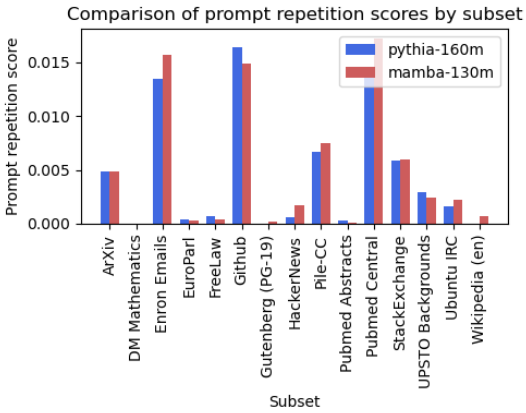
Aside from the data source, all other parameters such as number of samples and batch size remained consistent with the baseline setup, ensuring that the analysis of memorization patterns across different subsets was directly comparable to previous conditions.

3.4.2 Results and Analysis



(a) Fuzzy memorization

(b) Exact memorization



(c) Prompt repetition

Figure 3.6: Memorization and prompt repetition patterns measured on different subsets of The Pile dataset for memorization metrics: (a) fuzzy memorization, (b) exact memorization, and (c) prompt repetition.

The results indicate that both fuzzy and exact memorization scores follow a similar pattern, with the GitHub subset showing the highest memorization scores across both metrics. Notably, the Enron Emails subset also exhibits high memorization scores, particularly in fuzzy memorization, suggesting its conversational nature might contribute to its prominence. This is in contrast to the FreeLaw and HackerNews subsets, which have lower memorization scores.

Interestingly, while the fuzzy and exact memorization scores align, prompt repetition shows some divergence. GitHub, PubMed Central, and Enron Emails stand out with the highest levels of prompt repetition. Despite StackExchange demonstrating

higher memorization scores compared to PubMed Central in both exact and fuzzy memorization, it exhibits lower prompt repetition. This discrepancy highlights the variability in how different types of data influence memorization and prompt repetition patterns.

3.5 Input Prompt Length

This section examines how input prompt length affects memorization in transformer-based models. This also allows for a greater exploration of the role of attention mechanisms across different architectures. Recent research shows that attention dynamics are central to capturing long-range dependencies and can influence memorization [63]. Since attention mechanisms manage contextual information, variations in input length could reveal how attention is distributed among models of different sizes. Based on Carlini et al.'s findings [9], input length alone may not significantly influence memorization for transformers, but this has not been explored for SSMs. This study expands on this by evaluating whether varying input lengths impact memorization differently across model architectures.

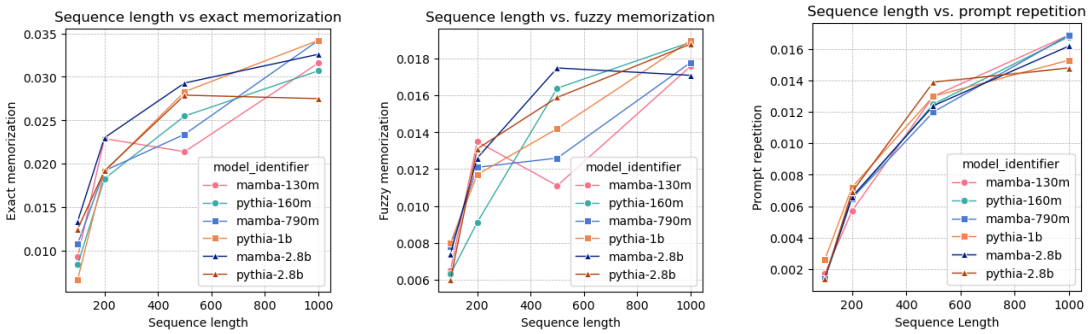
3.5.1 Experimental Procedure

To analyze the effect of input prompt length, three model sizes were selected from both the Mamba and Pythia suites trained on The Pile dataset: the smallest, intermediate, and largest models.

Data samples of varying lengths—100, 200, 500, and 1,000 tokens—were streamed from The Pile dataset. Each sample retained the last 50 tokens for evaluation, and used the rest as the input to the model. This setup was designed to assess whether longer prompts, which offer more context, affect memorization patterns differently across model architectures.

3.5.2 Results and Analysis

The results for exact memorization reveal that input length influences memorization patterns differently across models. While the transformer model Pythia aligned with the trends found in Carlini et al.'s research [9], showing memorization increasing across different sized input lengths, the exact memorization for the Mamba model is more complex. While Mamba exhibits similar trends for the shorter input lengths of



(a) Exact memorization (b) Fuzzy memorization (c) Prompt repetition

Figure 3.7: Results of memorization across different sized sequence lengths, which affect the size of input prompt to models. Metrics are shown for (a) exact memorization, (b) fuzzy memorization, and (c) prompt repetition.

sample sizes 100, 200, and 1,000, it provides a divergent result for sample length 500. Notably, the largest size Mamba model demonstrates expected findings, but the small and medium-sized models show slight inversions at this length. This divergence might be related to how attention mechanisms manage longer contexts in state space models, where attention behavior could differ significantly at higher input lengths, impacting memorization.

Interestingly, this finding is exacerbated when looking at the fuzzy memorization of the models. While the Pythia models show an increase in memorization up to length 500, the smallest Mamba model shows the greatest inversion at this length, with the medium showing slightly less inversion. These results suggest that model size affects how memorization scales with input length, indicating that larger models might handle longer contexts differently, possibly due to variations in attention dynamics.

Despite the divergence of results for Mamba within exact and fuzzy memorization, the results for both Pythia and Mamba models show similar patterns for prompt repetition. The pattern for prompt repetition memorization follows a similar pattern to Carlini et. al's results for exact memorization [9]. While not against the same metric, it is interesting to note that the type of memorization metrics shows that the Mamba acts similarly to the Pythia model in some of its output and differently for other metrics. These discrepancies may be influenced by dataset variance, yet both Pythia and Mamba models display increased prompt repetition for smaller models and longer input lengths. This warrants further investigation into the underlying causes and implications.

3.6 Summary and Insights

Across the various experiments, the role of prompt repetition initially appeared to offer little insight when examining model sizes. However, when considering subsets and input lengths, prompt repetition revealed deeper patterns of model memorization that were not as evident from other metrics. Notably, Mamba demonstrated greater memorization compared to the Pythia model. This observation was somewhat contradicted by the use of a sample size of 500 in the experiments. To gain a clearer understanding, it would be valuable to investigate larger Mamba models (e.g., 6.9 billion parameters) and to test a broader range of input lengths, between 200 and 1000, to identify if there is a specific length where memorization is minimal for smaller models.

Another significant finding was the higher memorization of GitHub content in both Pythia and Mamba models compared to other subsets. Further experiments could explore whether certain coding languages are more frequently memorised than others. Additionally, the DM Mathematics subset exhibited almost no memorization, which is intriguing given that code and mathematical content are often considered similar. This discrepancy suggests that different types of structured data may impact memorization differently.

Chapter 4

Performance Evaluation

In the previous chapter, we explored the prefix attack, examining how memorization can be exploited in models. Building on that foundation, we now shift our focus to evaluating the overall performance of these models on a variety of practical benchmarks. This chapter focuses on assessing how these models perform in tasks that reflect different domains of real-world challenges.

4.1 Purpose

Mamba has demonstrated stronger performance over transformers on a range of benchmarks [23, 26], including WinoGrande[51], ARC-C[14], ARC-E[14], PIQA[7], and HellaSwag[70]. Given these results, we anticipate that Mamba will achieve higher average scores on the LiveBench benchmarks compared to traditional transformers. Mamba's architecture, optimized for handling structured and sequential data through state-space dynamics, gives it a distinct edge in tasks requiring complex pattern recognition and analytical reasoning. In contrast, transformers, while highly effective in general-purpose tasks, may struggle with the analytical reasoning required in these benchmarks [63].

However, Mamba has shown greater memorization tendencies. Since LiveBench is uncontaminated—designed to assess performance on fresh, unseen data that was not in training—Mamba may score lower if its previous high scores were due to memorization rather than genuine capability. This raises a critical point: was Mamba's superior performance on prior benchmarks genuinely reflective of its strengths, or was it artificially inflated by memorization?

This chapter evaluates model performance across various benchmarks, focusing

on how memorization influences their effectiveness in real-world applications. The primary objective is to compare transformer models with the state space model (SSM) Mamba, highlighting their respective performance across diverse tasks.

4.2 Experimental Setup

To assess model performance beyond memorization, we used LiveBench, which provides six distinct benches: Data Analysis, Math, Coding, Instruction Following, Language, and Reasoning. These benchmarks contain multiple tasks that are regularly updated to reflect current standards and minimize biases from prior exposure, which aid in finding whether there is a tradeoff between memorization and performance between architectures.

Retrieval and Evaluation of Benchmarks

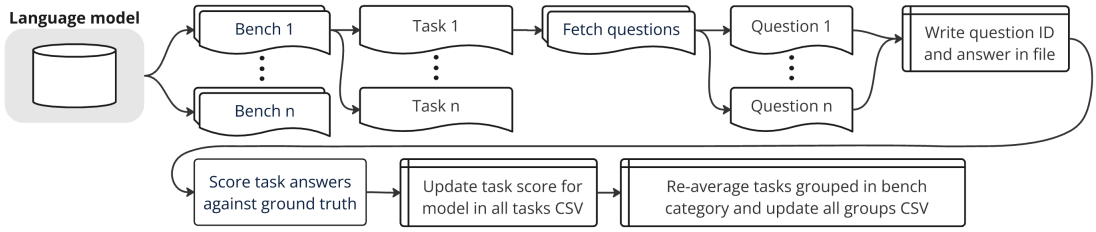


Figure 4.1: Diagram illustrating the workflow for benchmark evaluation and performance scoring. Bench, tasks, and questions are represented from 1 to n where 1 represents the first object and n lists the last object for each collection. The CSV file names 'all tasks' and 'all groups' correspond to the automatic files generated upon evaluation.

LiveBench ensures accurate and relevant question retrieval through an automated query system that is illustrated in Figure 4.1. Questions are selected based on recency and relevance to maintain alignment with current standards. Retrieved questions are organized and stored in a structured database, categorized by type, difficulty, and benchmark area, facilitating efficient access and updates.

The model's generated answers for each task's questions are saved in a corresponding JSONL file[1] containing question IDs, answer IDs, model IDs, choices, and

[1]A 'JSONL' file, or JSON Lines file, is a file format where each line is a separate JSON object. It is often used for storing large datasets where each line represents an individual data entry.

timestamps. As seen in Figure 4.1, these files are then evaluated with task scoring and bench group re-scoring.

4.3 Model Sizes and Variants

The primary focus of this section is to investigate the impact of model size on benchmark performance and to identify any emerging trends. Additionally, we then explore whether models of similar parameter sizes but trained on different datasets (The Pile vs. SlimPajama) exhibit comparable performance patterns.

4.3.1 Experimental Procedure

To address these questions, we first conducted experiments to compare the performance of the Pythia and Mamba models across five different sizes. For each model size, the models were tasked with generating answers for each bench's tasks. After all tasks were completed, the scores were gathered for each model.

Secondly, to explore models trained on different datasets, we evaluated the transformer-based BTLM-3b model and the SlimPajama Mamba model. For the BTLM model, a new adapter was created to ensure compatibility with Hugging Face and proper tokenizer setup, allowing it to be evaluated using the same methodology. Additionally, although updates were made to the LiveBench repository during the extension to the SlimPajama models—including new question sets and structural changes—these updates were excluded from this experiment to maintain consistency with prior benchmarks. While LiveBench aims to provide current benchmarks, we used the same dataset and questions as in previous experiments to ensure standardization and to facilitate direct comparisons of task breakdowns and category scores.

4.3.2 Performance Evaluation for Model Sizes

Table 4.1 presents the performance results across the categories, with specific analysis of each benchmark task comoposition scores in section D.2. When looking first to the **Math benchmark**, it is seen that Mamba models consistently outperform Pythia models for all model sizes. This trend is particularly noticeable when comparing the largest models, where Pythia scores 0.08 and Mamba scores 3.52. However, it is interesting to note that for this category the results do not indicate an increase with

model performance, as the smallest model is able to achieve a score of 4.15 and the largest decreases to 3.52.

Next, looking to the **Reasoning benchmark**, while the transformer models show a trend of scores being greater for larger models, Mamba models demonstrate high variability, as the highest score on this benchmark is for the model sized 370m and the largest 2.8b Mamba model scores zero. Further analysis of the tasks in this benchmark in Appendix D subsection D.2.6 demonstrate that the variation in scores is due to the Mamba model being only able to score on the second task, while the transformer model only able to score on the first task.

Table 2.1 also indicates no scores for **Coding benchmark** and **Data Analysis benchmark**. The evaluation producing scores of zero suggests that these tasks may require specific capabilities or fine-tuning that the evaluated models did not possess. However, low-scoring models of the leaderboard also showed scores of zero for this metric as seen in section D.2 Table D.1, which demonstrates the difficulty of this metric. Factors contributing to the no-score entries could include the complexity of the tasks, potential differences in benchmark design, and limitations of the models themselves. Additionally, the specific evaluation criteria of the benchmarks may have also played a role in the zero score.

The **Language benchmark** shows notably similar performance for the Pythia and Mamba models. While the Mamba is able to outperform the Pythia for the largest and smallest sizes, Pythia shows greater performance for mid-sized models. On the **Instruction Following benchmark** the Mamba model shows similar and slightly greater performance on smaller models. However, for larger models Pythia increasingly outperforms the Mamba model.

When analyzing the **Global Average** of the performance scores, it is evident that the models were not specifically fine-tuned for individual tasks, which could explain some of the observed variations. The Mamba-130m and Pythia-160m models exhibit similar average performance, with Mamba slightly outperforming Pythia. However, performance tends to deteriorate for larger Mamba models, particularly with the Mamba-1.4b and Mamba-2.8b models.

4.3.3 Performance Comparison to SlimPajama Models

After analyzing models trained on The Pile, we extended our comparison to models trained on the SlimPajama dataset to see if the trends maintained for architectures on a

Model	Global Average	Coding	Data Analysis	Math	Language	Reasoning	Instruction Following
Mamba-130m	**3.41**	0.00	0.00	**4.15**	**2.51**	1.00	**12.82**
Pythia-160m	3.26	0.00	0.00	4.00	2.44	1.00	12.15
Mamba-370m	**3.96**	0.00	0.00	**3.54**	1.47	**5.00**	**13.77**
Pythia-410m	2.85	0.00	0.00	1.76	**2.31**	1.00	12.05
Mamba-790m	**3.54**	0.00	0.00	**2.31**	2.03	4.00	12.91
Pythia-1b	3.23	0.00	0.00	0.00	**2.12**	4.00	**13.25**
Mamba-1.4b	3.92	0.00	0.00	**4.45**	2.01	1.00	16.07
Pythia-1.4b	**4.71**	0.00	0.00	2.12	**3.68**	**4.00**	**18.43**
Mamba-2.8b	2.60	0.00	0.00	**3.52**	**2.19**	0.00	9.88
Pythia-2.8b	**3.19**	0.00	0.00	0.08	2.06	**3.00**	**13.98**
Mamba-2.8b-SP	2.22	0.00	0.00	0.44	**2.34**	0.00	10.50
BTLM-3b-SP	**6.68**	0.00	**0.19**	**0.90**	1.88	**12.00**	**25.09**

Table 4.1: Performance of models on global average of all categories and the results of each benchmark category from LiveBench: Coding, Data Analysis, Math, Language, Reasoning, and Instruction Following. Bolded values represent the highest result within each benchmark category, with bolded models indicating highest overall average. The bottom two models, denoted with the `-SP` suffix, were trained on the SlimPajama dataset, while the other models were trained on The Pile. Scores reflect the automatic evaluation results from LiveBench.

different dataset.

Overall, the trends exhibited for the largest size Mamba and Pythia model maintain for the Mamba-SP and BTLM-SP models. The only category that does not follow this trend is for the Math benchmark, where the Mamba performs noticeably lower than it had previously and the transformer BTLM is able to score higher. Additionally, the BTLM transformer model was able to achieve a score for the Data Analysis benchmark that had previously only acheived zero scores. This suggests that the training data may be an important factor for models being able to score on this metric.

Further comparison of the SlimPajama against The Pile models reveal that the SlimPajama dataset allowed the transformer model to score four times greater for the Reasoning benchmark and almost double for the Instruction Following benchmark when compared to The Pile transformer counterpart. For these metrics, the Mamba showed negligible difference.

Overall, the comparative analysis between models trained on The Pile and SlimPajama datasets reveals that while Mamba models perform strongly in certain benchmarks like Math, transformer models maintain superior performance in reasoning and instruction-following tasks for large model sizes. These insights underline the influence of training data and model architecture on overall performance.

4.4 Summary and Insights

The Mamba models exhibit clear advantages in math-related benchmarks compared to transformers, highlighting the effectiveness of the state space model architecture in managing numerical and pattern-based tasks. Despite this strength, there is notable variability in Mamba's performance on reasoning tasks, with larger models showing reduced effectiveness. This suggests that while Mamba excels in mathematical problem-solving, its capabilities in reasoning and instruction-following require further investigation or fine-tuning.

Additionally, Mamba's underperformance in data analysis and coding tasks raises questions about its generalizability across different benchmarks. This discrepancy could indicate that the model struggles with more complex tasks or that the benchmarks used were particularly challenging. Further analysis should explore whether Mamba's lower performance is due to the inherent difficulty of these benchmarks or limitations in the model itself, by evaluating its ability to produce nonzero scores on simpler coding and data analysis tasks. Including transformer models in this investigation would provide a clearer understanding through comparison.

Notably, Mamba's performance decline on tasks where it previously excelled suggests that earlier high scores may have been influenced by memorization rather than genuine model capabilities. To address this, a detailed analysis is needed to determine if memorization of specific training data contributed to these discrepancies.

The evaluation of models trained on the SlimPajama dataset reinforces the significant impact of training data and model size on performance. Future research should focus on fine-tuning models for specific tasks to improve their performance to see if that could improve its performance. Additionally, exploring adjustments in training data and model architecture may help overcome current limitations. Overall, while Mamba demonstrates strong performance in certain benchmarks, ongoing research is essential to address its limitations and enhance its capabilities across a wider range of tasks.

Chapter 5

Membership Inference

This chapter looks more closely into the memorization of the architectures through the lens of membership inference attacks (MIAs) – an attack that examines the likelihood of a specific data sample being included in the model's training dataset. This section compares the architectures by exploring the performance of the models across MIA attacks for methods of Neighborhood Attack, Min-K%, and Min-K%++. It also explores the metrics across model variants, different-sized models, and different input lengths.

5.1 Purpose

While Mamba has shown greater memorization when evaluated on prefix adversarial attacks, it was still able to outperform transformer models for mathematical tasks. This suggests that though it has demonstrated increased memorization of passages of training data, it may be a worthwhile choice depending on the purpose of the model. Expanding on the initial work done on memorization to MIA attacks allows for a greater assessment of the susceptibility to different styles of attacks by being able to generate metrics for the likelihood that the model will be able to identify data it has been trained on.

Although a different style of attack, due to the high memorization observed for prefix attacks, suggests that the Mamba model may also show heightened susceptibility to MIAs compared to Pythia. If models that performed well in memorization tasks are vulnerable to membership inference, it would suggest that their ability to retain and recall training data could be compromising their privacy [28].

MIA attacks were also chosen as they often lay the groundwork for defense mechanisms for unlearning as they help identify which data points are memorized by the model. Through this, data can be classified to find the best candidates for unlearning,

which has shown to be successful in improving the model's privacy and robustness against MIA vulnerabilities [12, 54]. In using this attack, this experiment can better offer insight into the best future direction for unlearning to apply defenses to the models.

This chapter aims to provide a comprehensive evaluation of membership inference risks associated with the models studied, using the observed memorization patterns to gauge privacy vulnerabilities and highlight potential avenues for future privacy enhancements.

5.2 Experimental Setup

5.2.1 Dataset

Because the dataset for membership inference attacks includes data used in training and similar data not used in training, allows for the attacks to assess the likelihood of the seen versus unseen data. Instead of sourcing The Pile and non-The Pile data for this experiment, this study utilizes a benchmarked membership inference dataset WikiMIA [56]. WikiMIA was selected because it offers different-sized input lengths (32, 64, 128, and 256 tokens), which are useful for examining how models handle different text lengths. Additionally, because the set is benchmarked, it allows the results of this study to not only be reproducible but also allows the findings to be compared to other MIA studies. This choice enables an analysis of the impact of input length on membership inference.

MIA attacks do this by calculating the score of the likelihood for the input given to be included in the training data. In methods like the Neighborhood Attack and Min-K% attacks, thresholds are used to classify samples based on their loss or probability characteristics. For instance, in the Neighborhood Attack, a threshold might be set to decide if the loss of a target sample significantly deviates from its neighbors. Similarly, Min-K% and Min-K%++ use thresholds to classify tokens or sequences based on their probabilities. While the application of thresholds differs, the underlying concept of setting criteria to classify samples is similar across these methods.

5.2.2 Neighborhood Attack

The Neighborhood Attack method, as shown in Figure 5.1, infers membership by comparing the loss of a target sample with the losses of its syntactically and semantically similar neighbors. This technique operates under the assumption that if a sample was

included in the training data, its loss would be comparable to that of its neighbors. For this study, we employed the BERT-base-cased model [16] for the proposal model that would create the neighbor samples as it aligns with prior research done for the neighborhood attack [37, 67]. To evaluate the membership inference, we evaluated 10 neighbors against each target sample, as this number aligned with established practices in the literature [37], and allowed for a meaningful comparison while maintaining manageable computational requirements.

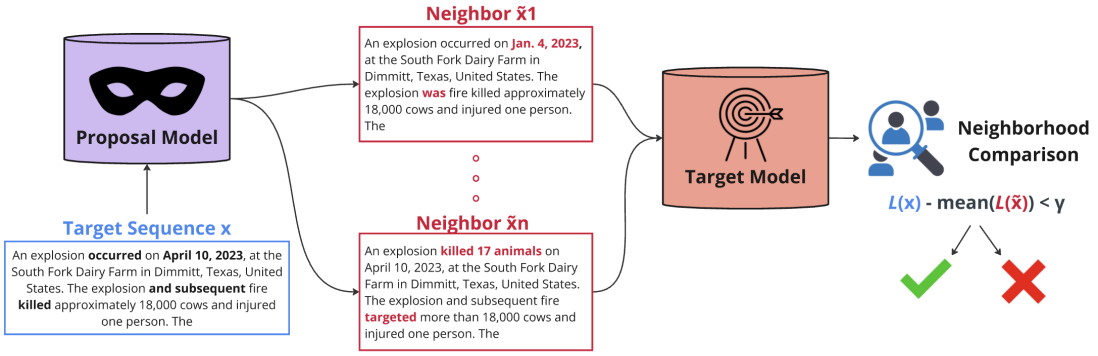


Figure 5.1: For a given target sample x, a pretrained masked language model generates a set of neighbor sentences by substituting words in the original sample. The losses of these generated neighbors and the original sample are then compared using the target model. If the losses for the neighbors are similar to the loss for the target sample, and the difference between them is below a specified threshold γ, this suggests that the target sample is likely to be part of the model's training data.

5.2.3 Min-K% and Min-K%++ Methodologies

The **Min-K%** and **Min-K%++** methods are used to determine whether a text sequence was part of a model's training data by analyzing token probabilities. These methods are both scoring functions, with process illustrated in Figure 5.2.

The **Min-K%** method focuses on tokens with the lowest probabilities within a sequence. It calculates the average log-likelihood of the lowest $k\%$ of these tokens. This approach works on the assumption that tokens from non-member sequences often exhibit unusually lower probabilities, making them easier to identify. The log-likelihood and corresponding score for this method is computed using the following formula:

$$\text{Min-K\%}_{\text{token}}(x_t) = \log p(x_t \mid x_{<t}), \tag{5.1}$$

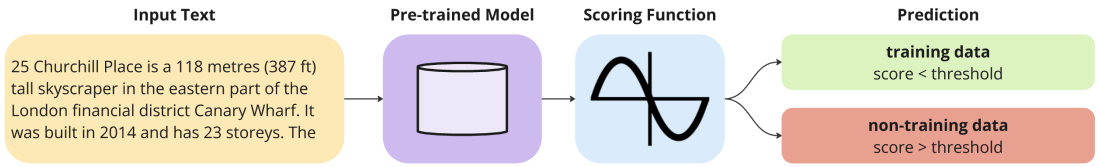


Figure 5.2: Process of evaluation for both the the Min-K% and Min-K%++ methodologies. Both methods are used for scoring is able to determine whether the text that the model was prompted with was included in the training data.

$$\text{Min-K\%}(x) = \frac{1}{|\text{min-K\%}|} \sum_{x_t \in \text{min-K\%}} \text{Min-K\%}_{\text{token}}(x_t). \tag{5.2}$$

where $p(x_t \mid x_{<t})$ represents the probability of token x_t given the preceding tokens $x_{<t}$. The overall Min-K% score is then calculated by averaging these log-likelihood values over the minimum-K% of tokens.

The **Min-K%++** method extends Min-K% by introducing a normalization step to reduce the impact of variability in token probabilities. This method calculates the log probability of each token, but then normalizes this value by subtracting the mean and dividing by the standard deviation of the log probabilities for all candidate tokens. The normalized log probability and given score is given by:

$$\text{Min-K\%++}_{\text{token}}(x_t) = \frac{\log p(x_t \mid x_{<t}) - \mu_{x_{<t}}}{\sigma_{x_{<t}}}, \tag{5.3}$$

$$\text{Min-K\%++}(x) = \frac{1}{|\text{min-K\%}|} \sum_{x_t \in \text{min-K\%}} \text{Min-K\%++}_{\text{token}}(x_t). \tag{5.4}$$

where $\mu_{x_{<t}}$ and $\sigma_{x_{<t}}$ are the mean and standard deviation of the log probabilities, respectively.

It can be seen from Equation 5.2 and Equation 5.4 that these methodologies are structured similarly, with the only difference being the normalization of tokens of Equation 5.1, which can be seen in Equation 5.3. While Min-K% has proven effective for evaluating membership inference and unlearning, this research incorporates the newer Min-K%++ method because of the strong findings. Min-K%++ introduces $\sigma_{x<t}$, inspired by temperature scaling [25], to calibrate prediction confidence by adjusting model outputs with a dynamic, input-adaptive factor rather than a fixed constant. This approach has shown to improve performance, especially in cases with variable token distributions. By normalizing log probabilities, the Min-K%++ research aims to provide a more balanced and accurate assessment compared to the simpler Min-K% method [71].

Because of the similarities, both will be evaluated against the architectures to examine if the differing architectures of comparable sizes present similar patterns across the evaluation metrics.

5.2.4 Output Results

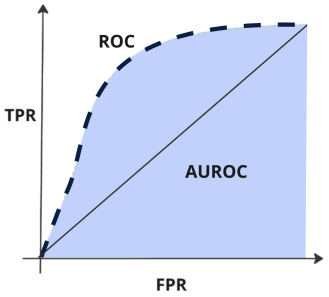


Figure 5.3: Receiver Operating Characteristic (ROC) curve is created from plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). This curve illustrates how well the model distinguishes between training data members and non-members. The Area Under the ROC Curve (AUROC) represents the overall effectiveness of the model, with higher values indicating better performance in identifying whether a sample was part of the training data.

For evaluating membership inference attacks, we used consistent rates across all models. Specifically, a True Positive Rate (TPR) threshold of 95% was set, meaning the model aims to correctly identify 95% of actual members. This threshold automatically determines the False Positive Rate (FPR) as 5%, indicating the proportion of non-members incorrectly classified as members. This approach balances sensitivity and specificity, as shown in Figure 5.3 and uses values that are in line with standard practices for evaluating membership inference attacks [67, 57, 10].

5.2.5 Limitations

Although the WikiMIA dataset contains sizes of input up to 256 tokens, later experiments for input length only tested the lower three input sizes of 32, 64, and 128 tokens due to time constraints. Additionally, these experiments were limited to the WikiMIA dataset to focus on input length, but further experiments could expand to different subsets of The Pile through testing with the other benchmarked membership inference dataset MIMIR [18], which contains subsets of MIA data collection for subsets of The

Pile. This was not done due to the time needed to evaluate each subset, however, it would help expand on the initial work done for prefix attacks on subsets. This direction would allow for a greater understanding of security for specific domains.

5.3 Neighborhood Attacks

The goal of this section is to evaluate how the SSM and transformer architectures exhibit memorization based on their AUROC score on the neighborhood attack, and whether a smaller model and larger model will show similar patterns. Additionally, experiments were done to expand the study to transformer and transformer-like models of different sizes to broaden the study, as was done previously for prefix attacks.

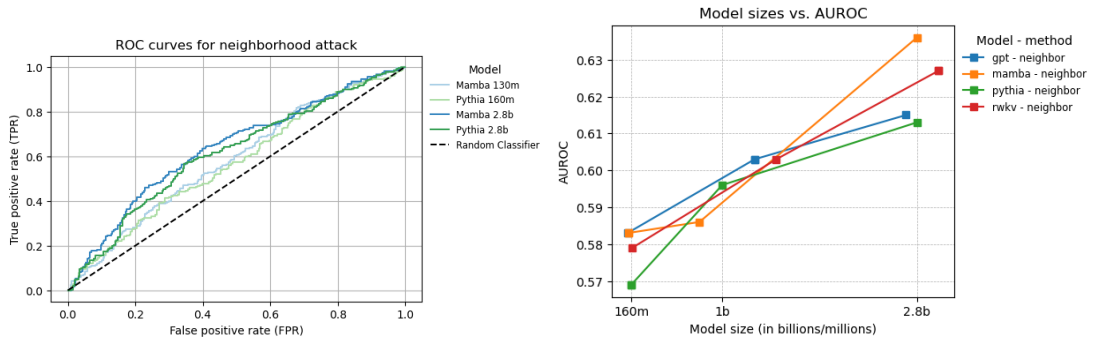
5.3.1 Experimental Procedure

To address the SSM and transformer architecture on the Neighborhood Attack, we first selected the smallest and largest models from the Mamba and Pythia suites. They were then evaluated on 10 generated neighbor samples and compared to the initial sample. Each model was subjected to the attack from the WikiMIA sample with an input length of 64, which was chosen as the (something about 64 is greater than 32 but less than 128 so it is a central metric).

This evaluation was then extended to include RWKV-4 and GPT-Neo models as well as including the small, medium, and large variants for each model type. Each model was attacked on the same set of samples and 10 neighbors for standardization.

5.3.2 Model Sizes

The results of this experiment indicate that Mamba shows greater memorization as indicated by the larger ROC blue curves for the Mamba model compared to the green curves for the Pythia in Figure 5.4a. Further looking at the lighter colored curves that are drawn for the smaller architectures against the darker colored curves for the larger in Figure 5.4a, demonstrates a similar increase in memorization across small to large model sizes for both Mamba and Pythia.



(a) (b)

Figure 5.4: (a) Shows the TPR against the FPR for the smallest and largest Pythia and Mamba models, which create the ROC curve on an input length of 64. (b) Shows the measurement of different models, and their respective sizes against their corresponding AUROC score on an input length of 64.

5.3.3 Additional Models

Expanding on the initial findings of the comparison between the small and large Mamba and Pythia models on the Neighborhood Attack, other models with transformer and transformer-like architectures were examined to see the effect of model size on the memorization which can be viewed in Figure 5.4b. When observing the line graph, transformer-based architectures GPT-Neo and Pythia perform similarly, with the Pythia model showing less memorization for the smaller model size.

The greatest divergence of the memorization metrics occurs for the larger sizes of Mamba and the RNN-transformer hybrid RWKV-4. These demonstrate greater memorization than the transformer models. The high metric for RWKV-4 suggests that the previous low memorization performance of the RWKV-4 model on the prefix attack was due to the difficulty in tokenizing the longer input. It is also notable that the hybrid RNN appears to exhibit higher memorization than the pure transformer models, but still less than the SSM architecture of Mamba.

5.4 Comparison of Membership Inference Attacks

This section examines the performance of membership inference attacks across model sizes and input lengths to see if there are similar trends to those found for the prefix attacks. The goal is to assess how sample input length and model size influence membership inference across all metrics to identify any common trends. This analysis

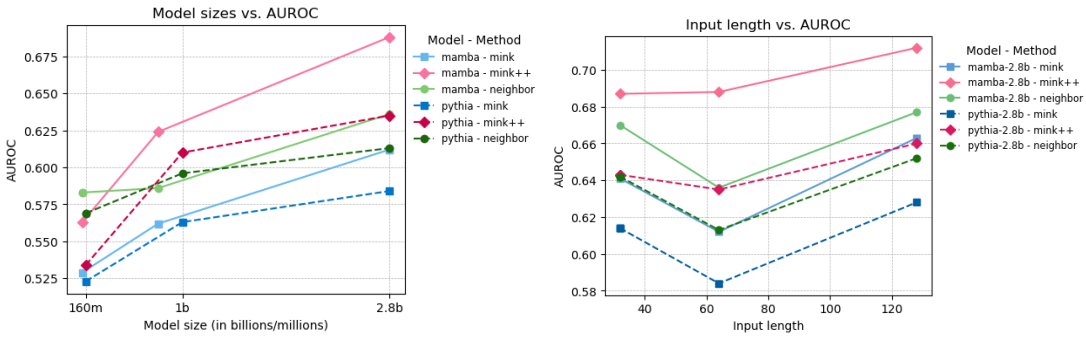
parallels similar experiments done previously for prefix attacks to determine if they exhibit comparable effects.

A more detailed analysis to compare the Min-K% and Min-K%++ methods on the influence of k on AUROC values for the 2.8b Pythia and Mamba models was also explored and can be found in Appendix B section B.4.

5.4.1 Experimental Procedure

For testing the effect of the model size across the three attacks, the small, large, and extra large were chosen for Pythia and Mamba models. These were tested on WikiMIA lengths of 64. Within the Neighborhood Attack, 10 neighbors were used for comparison. Next, for testing the WikiMIA length, lengths 32, 64, and 128 were tested on the largest Pythia and Mamba models. The 2.8b models were chosen for analysis as it showed the largest models demonstrated the most divergence in the previous experiment(Figure 5.4b), and thus might be easiest to visualize the divergent patterns between the architectures.

5.4.2 Effect of Model Size



(a) (b)

Figure 5.5: (a) AUROC scores of Min-K% and Min-K%++ methods across different model sizes to demonstrate memorization. (b) Effect of input length on Min-K% and Min-K%++ performance of classification, demonstrated by the AUROC score.

Figure 5.5a displays the performance of the MIA methods across different model sizes. Most notable is as size increases for the Mamba and Pythia models, on the Neighborhood Attack in green is the slightly lower score for Mamba. This differs from the other MIA attacks, where Mamba, in solid lines, consistently shows greater

memorization than its dashed counterpart. The overall trends of the graph indicate that for scoring, the MIA Min-K%++ attack demonstrates the greatest AUROC scores, followed by the Neighborhood Attack, with the Min-K% having the lowest scores. However, for the smallest models, the Neighborhood Attack demonstrates the most effective at getting both Pythia and Mamba models to achieve the highest AUROC scores. This suggests that for smaller models the Neighborhood Attack may be more effective, but for larger models, the Min-K%++ could be more effective.

5.4.3 Effect of Input Length

When comparing the results for input lengths of 32 and 64 tokens, there is a notable decrease in AUROC scores for all models and MIA attacks(Figure 5.5b). Interestingly, the decrease appears smallest for the Mamba model in the Min-K%++ attack and greatest for the Mamba model in the Neighbor attack. This suggests that depending on the size of the input sequence, there could be more optimal MIA attacks. Across all MIA attacks the Mamba is shown to memorize more than its Pythia counterpart, which reflects similar results to the previous experiment on model size.

5.5 Summary and Insights

Evaluation of the variety of MIA attacks demonstrated that Mamba had greater memorization, as predicted by the AUROC scores across Min-K%, Min-K%++, and Neighborhood Attack. While demonstrating less memorization than Mamba, the RWKV-4 exhibited interesting behavior by showing greater memorization for larger-sized models than the transformer models. Additionally, The findings across model sizes indicate that the Neighborhood attack for mid-sized Mamba and Pythia models should be further examined as Mamba showed less performance. Lastly, when examining the effect of the input length, the results for size 64 appeared the smallest. As the base experiment for the experiments conducted in this chapter, similar experiments of sizes 32 and 128 could be done to further assess if the results are a novelty of the 64 lengths or whether they are consistent. Additionally, adding the input length of 256 could help evaluate the trend of input length and may highlight the patterns better.

Chapter 6

Conclusions

6.1 Summary

This work focused on analyzing the memorization and performance of SSM and transformer architectures by comparing the Mamba (SSM) and Pythia (transformer) models. The initial exploration focused on analyzing memorization using prefix attacks, where the model was given the start of a training data passage and its output was compared to the actual end of that passage. The findings showed that Mamba exhibited greater exact and fuzzy memorization, consistent with previous research suggesting a correlation between Mamba's superior recall in n-gram prefix lookup tasks and its performance in these memorization tests [31]. Further analysis revealed higher memorization for both models on the GitHub subset compared to other subsets of The Pile. Additionally, when examining input lengths for larger samples of 500 tokens, the Mamba model displayed divergent behavior, possibly due to varying attention mechanisms across different Mamba model sizes. Next, evaluations on the uncontaminated LiveBench benchmark aligned with previous research [23, 26] for smaller models, with Mamba averaging higher scores than Pythia. However, for larger models, the results deviated, with Pythia outperforming Mamba on these tasks. Mamba models generally excelled in mathematical tasks, while transformers performed better in instruction-following tasks. Notably, neither model scored on the data analysis and coding tasks, likely due to the benchmarks' difficulty. Lastly, memorization was revisited in the domain of membership inference, which can determine the likelihood that a piece of data was shown in the model's training. The findings across multiple MIA attacks demonstrated greater memorization for Mamba models, which supported the previous findings for the prefix attacks.

6.2 Limitations

The greatest limitation within this study was the availability of pre-trained Mamba models. As larger models demonstrate are shown to exhibit greater memorization with memorization examined on models of sized 6.9b and greater [39], the trends found for this study may not scale the same for the Mamba architecture as they do for the Pythia architecture. Additionally, because of the divergent architectures, pre-trained models on de-duped and quantized models would allow for greater ability to measure the effect of those as defences [33, 36] across the different architectures in being able to analyze the strongest defense technique per architecture.

Another limitation of this study was the focus on generative models, such as Pythia and Mamba, did not allow for other adversarial attacks such as masked attacks that require fill-mask functionality [73]. This constraint prevented the analysis of masked attacks on these models. The study would have also benefited from further evaluation of the models on easier performance benchmarks for coding and data analysis tasks to better understand the performance difference between the architectures. However, such an analysis was not feasible due to computational constraints.

6.3 Future directions

Based on the initial experiments with prefix attacks, a more detailed analysis of memorization on 500-token samples across different sizes of the Mamba model is needed. This should include how it performs for tasks that have a prompt of this length as well as assessing the variation of memorization across the subsets of The Pile to assess the distribution of membership inference across the subsets. However, the results suggest that this could be due to attention, and should be explored further. Additionally, further work on analyzing the comparative performance of the models on less difficult coding and data analysis tasks, such as Codereval [68] and InfiAgent-DABench [30], is needed to better understand whether the 0 score was due to both models' inability to perform these styles of attacks, or if, on simpler tasks, one of the models is capable of superior performance. Lastly, additional work should be done to expand on the initial MIA work to determine the effect unlearning has on the models. Specifically, if for the same setup, the Mamba model is more effective at unlearning and retaining performance compared to transformer models, which have been shown to be effective at unlearning by targeting data identified as likely memorized through MIA [54].

Bibliography

[1] Together AI. Mamba 3b and slimpajama: Scaling language models and datasets, 2024.

[2] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifiers. *ICLR*, 2016.

[3] Markus Bayer, Markus Neiczer, Maximilian Samsinger, Björn Buchhold, and Christian Reuter. Xai-attack: Utilizing explainable ai to find incorrectly learned patterns for black-box adversarial example creation. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 17725–17738, 2024.

[4] Yoshua Bengio. Learning deep architectures for ai. In *Foundations and Trends® in Machine Learning*, volume 2, pages 1–127. Now Publishers Inc, 2013.

[5] Stella Biderman, Kieran Bicheno, and Leo Gao. Datasheet for the pile. *arXiv preprint arXiv:2201.07311*, 2022.

[6] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[7] Yejin Bisk, Carlos C. H., and Robyn Zhang. Piqa: Reasoning about physical interactions with objects. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*. Association for Computational Linguistics, 2020.

[8] Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021.

[9] Nicholas Carlini, Luke Melis, Iulian Serban, et al. Extracting training data from large language models. In *USENIX Security Symposium*, 2021.

[10] Nicholas Carlini and David Wagner. Membership inference attacks against machine learning models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2019.

[11] Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural networks: An analysis. *Journal of Machine Learning Research*, 2:1–22, 2001.

[12] A. Carvalho, A. Nascimento, and A. Cramer. Machine learning unlearning: A survey of unlearning techniques. *ACM Computing Surveys*, 52(4):1–24, 2019.

[13] Shan Chen, Jack Gallifant, Mingye Gao, Pedro Moreira, Nikolaj Munch, Ajay Muthukkumar, Arvind Rajan, Jaya Kolluri, Amelia Fiske, Janna Hastings, et al. Cross-care: Assessing the healthcare implications of pre-training data on language model bias. *arXiv preprint arXiv:2405.05506*, 2024.

[14] Christopher Clark and Matt Gardner. Think you have solved reading comprehension? try arc, the ai2 reasoning challenge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*. Association for Computational Linguistics, 2018.

[15] Together Computer. Redpajama: an open dataset for training large language models, 2023.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[17] Nolan Dey, Daria Soboleva, Faisal Al-Khateeb, Bowen Yang, Ribhu Pathria, Hemant Khachane, Shaheer Muhammad, Robert Myers, Jacob Robert Steeves, Natalia Vassilieva, et al. Btlm-3b-8k: 7b parameter performance in a 3b parameter model. *arXiv preprint arXiv:2309.11568*, 2023.

[18] Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. Do membership inference attacks work on large language models? *arXiv preprint arXiv:2402.07841*, 2024.

[19] Daniel Y Fu, Tri Dao, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

[20] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[21] Adam Goodkind and Klinton Bicknell. Predictive power of word surprisal for reading times is a linear function of language model quality. In *Proceedings of the 8th workshop on cognitive modeling and computational linguistics (CMCL 2018)*, pages 10–18, 2018.

[22] Riccardo Grazzi, Julien Siems, Simon Schrodi, Thomas Brox, and Frank Hutter. Is mamba capable of in-context learning? *arXiv preprint arXiv:2402.03170*, 2024.

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

[24] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[25] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[26] John T Halloran, Manbir Gulati, and Paul F Roysdon. Mamba state-space models can be strong downstream learners. *arXiv preprint arXiv:2406.00209*, 2024.

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS*, 2015.

[28] B. Hitaj, G. Ateniese, and J. Spillner. Deep models under the gan: Information leakage from gradients. In *2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 603–618. ACM, 2017.

[29] Sanghyun Hong, Michael-Andrei Panaitescu-Liess, Yigitcan Kaya, and Tudor Dumitras. Qu-anti-zation: Exploiting quantization artifacts for achieving adversarial outcomes. *Advances in Neural Information Processing Systems*, 34:9303–9316, 2021.

[30] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

[31] Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024.

[32] S Jeremy. *Einstein's Riddle: Riddles, Paradoxes, and Conundrums to Stretch Your Mind*. Bloomsbury USA, 2009.

[33] Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. In *International Conference on Machine Learning*, pages 10697–10707. PMLR, 2022.

[34] Antonia Karamolegkou, Jiaang Li, Li Zhou, and Anders Søgaard. Copyright violations and large language models. *arXiv preprint arXiv:2310.13771*, 2023.

[35] Phil Katz. File compression. In *ACM SIGMOD Record*, volume 21, pages 46–56. Association for Computing Machinery, 1992.

[36] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*, 2019.

[37] Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schölkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership inference attacks against language models via neighbourhood comparison. *arXiv preprint arXiv:2305.18462*, 2023.

[38] Avijit Mitra, Emily Druhl, Raelene Goodwin, and Hong Yu. Synth-sbdh: A synthetic dataset of social and behavioral determinants of health for clinical text. *arXiv preprint arXiv:2406.06056*, 2024.

[39] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian

Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023.

[40] Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Michael Wornow, Callum Birch-Sykes, Stefano Massaroli, Aman Patel, Clayton Rabideau, Yoshua Bengio, et al. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *Advances in neural information processing systems*, 36, 2024.

[41] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Nghia The Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, 2016.

[42] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318. Association for Computational Linguistics, 2002.

[43] R. Parikh, C. Dupuy, and R. Gupta. Canary extraction in natural language understanding models. *arXiv preprint arXiv:2203.13920*, 2022.

[44] Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks. *arXiv preprint arXiv:2402.04248*, 2024.

[45] Rupesh Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318. JMLR.org, 2013.

[46] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only. 2023.

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

[48] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.

[49] Adam Roberts, Colin Raffel, Katherine Lee, Michael Matena, Noam Shazeer, Peter J Liu, Sharan Narang, Wei Li, and Yanqi Zhou. Exploring the limits of transfer learning with a unified text-to-text transformer. *Google, Tech. Rep.*, 2019.

[50] Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. Nlp evaluation in trouble: On the need to measure llm data contamination for each benchmark. *arXiv preprint arXiv:2310.18018*, 2023.

[51] Claudia Schmaus, Jason Davidson, and Stephen Mo. Winogrande: A large-scale benchmark for coreference resolution. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*. Association for Computational Linguistics, 2020.

[52] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[53] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[54] Rohan Sharma, Shijie Zhou, Kaiyi Ji, and Changyou Chen. Discriminative adversarial unlearning. *arXiv preprint arXiv:2402.06864*, 2024.

[55] Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*, 2023.

[56] Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789*, 2023.

[57] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.

[58] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[59] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[60] J.-B. Truong, P. Maini, R. J. Walls, and N. Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4771–4780, 2021.

[61] Dušan Variš and Ondřej Bojar. Sequence length is a domain: Length-based overfitting in transformer models. *arXiv preprint arXiv:2109.07276*, 2021.

[62] Apostol Vassilev, Alina Oprea, Alie Fordyce, and Hyrum Anderson. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. Technical Report NIST AI 100-2e2023, National Institute of Standards and Technology (NIST), 2023.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[64] Jeffrey G Wang, Jason Wang, Marvin Li, and Seth Neel. Pandora's white-box: Increased training data leakage in open llms. *arXiv preprint arXiv:2402.17012*, 2024.

[65] Siyuan Wang, Zhuohan Long, Zhihao Fan, Zhongyu Wei, and Xuanjing Huang. Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation. *arXiv preprint arXiv:2402.11443*, 2024.

[66] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum.

Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

[67] Seongju Yeom, Iacopo Giacomelli, Seungjin Oh, and Sanmi Koyejo. Privacy risks across ML models: A case study on membership inference. In *Proceedings of the 2018 ACM Conference on Computer and Communications Security (CCS)*, pages 89–106. ACM, 2018.

[68] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–12, 2024.

[69] Ling Yue, Sixue Xing, Yingzhou Lu, and Tianfan Fu. Biomamba: A pre-trained biomedical language representation model leveraging mamba. *arXiv preprint arXiv:2408.02600*, August 2024. Submitted on 5 Aug 2024.

[70] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019.

[71] Jingyang Zhang, Jingwei Sun, Eric Yeats, Yang Ouyang, Martin Kuo, Jianyi Zhang, Hao Yang, and Hai Li. Min-k%++: Improved baseline for detecting pre-training data from large language models. *arXiv preprint arXiv:2404.02936*, 2024.

[72] Z. Zhang, J. Wen, and M. Huang. Ethicist: Targeted training data extraction through loss smoothed soft prompting and calibrated confidence estimation. *arXiv preprint arXiv:2307.04401*, 2023.

[73] He Zhu, Ce Li, Haitian Yang, Yan Wang, and Weiqing Huang. Prompt makes mask language models better adversarial attackers. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

Appendix A

Architectures

A.1 RWKV Architecture

RWKV models leverage a combination of recurrent mechanisms and weighted key-value memory to handle sequential data. Their design and components are tailored to capture and memorize patterns effectively. The diagram of the components can be seen in Figure A.1, with the key aspects of RWKV models being:

1. **Recurrent Weighted Key-Value Memory**: The RWKV architecture incorporates a recurrent mechanism with a key-value memory component. This mechanism maintains a dynamic memory of past information, where each token in the sequence is associated with a set of weighted key-value pairs. The recurrence enables the model to retain and use information from previous tokens, which is crucial for understanding long-range dependencies in the data. The memory update process involves:

$$\text{Memory}_{t+1} = \text{Update}(\text{Memory}_t, \text{Input}_t)$$

   where Update represents a function that combines the previous memory with the current input. This setup allows the model to store and recall information over extended sequences, making it adept at capturing and utilizing patterns observed in the data.

2. **Recurrent Processing with Weighted Key-Value Storage**: RWKV models process inputs recurrently, applying transformations to the input tokens and their associated memory:

$$\text{Output}_t = \text{Transform}(\text{Input}_t, \text{Memory}_t)$$

where Transform represents the function that computes the output based on the input and the current memory. This approach helps the model generate outputs by leveraging both the current input and the stored context, allowing it to capture complex patterns and relationships in the data.

3. **Scalable Memory Management and Efficiency**: RWKV models are designed to be efficient in handling large-scale data. The architecture balances the need for comprehensive memory with computational efficiency:

$$\text{Efficient Memory Management} = \text{Scale}(\text{Memory}, \text{Data Size})$$

where Scale adjusts the memory capacity according to the size of the input data. This feature ensures that the model can handle long sequences and large datasets effectively, making it scalable for various applications.

4. **Integration of Recurrence with Transformer-like Components**: While RWKV models utilize recurrent mechanisms, they also incorporate components inspired by transformer architectures, such as attention mechanisms. This hybrid approach enables the model to benefit from both recurrent and attention-based processing:

$$\text{Hybrid Processing} = \text{Recurrent Processing} + \text{Attention Mechanisms}$$

This integration allows RWKV models to leverage the strengths of both recurrent and attention-based methods, improving their ability to capture and generate patterns in data.

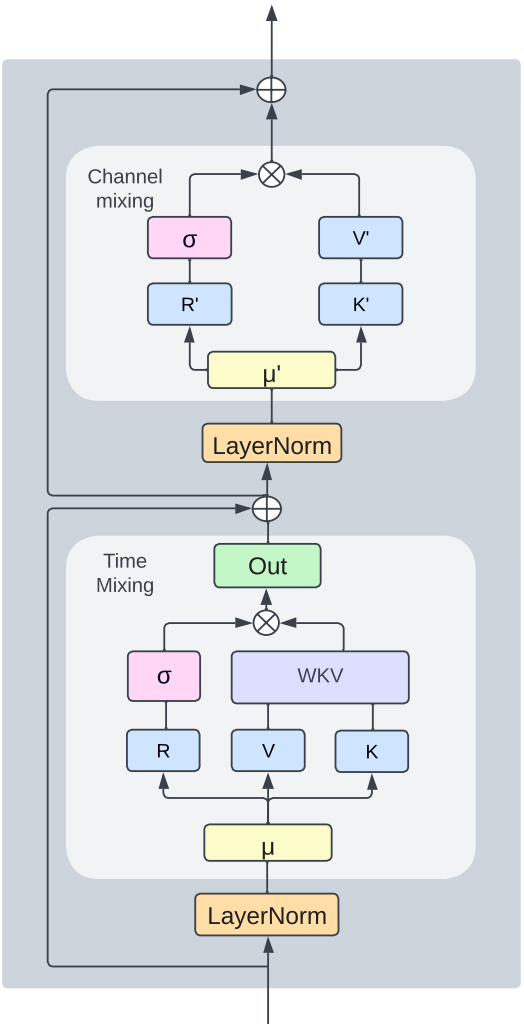


Figure A.1: Architecture of the RWKV model, illustrating the integration of RNN and transformer components.

Appendix B

Additional Membership Inference

B.1 Perplexity

Perplexity is a common metric used to evaluate language models and their ability to predict a sample of text. It is defined as the exponentiation of the entropy, which measures the average uncertainty in predicting the next token. For a given model, the perplexity PP of a sequence of tokens is calculated as:

$$\mathrm{PP} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right)$$

where $p(x_i)$ is the probability assigned by the model to token x_i, and N is the total number of tokens. Lower perplexity values indicate that the model is better at predicting the sequence, reflecting a more accurate representation of the language structure.

The concept of perplexity originates from information theory and was introduced to the field of natural language processing as a way to assess the performance of probabilistic models. The metric was initially used by researchers like Shannon in the 1940s to measure the uncertainty in predicting sequences of symbols [53]. In modern NLP, perplexity provides insight into how well a model captures the statistical properties of the language [21].

While perplexity can be a measure of membership inference, other studies on the perplexity for response to LAMBADA questions by measuring the responses of perplexity demonstrated lower perplexity scores for each Mamba model than to each transformer counterpart [23, 26]. Despite studies from Calini using perplexity as a metric [9] for a partial relation to the model per

B.2 Zlib Compression Score

The zlib compression score measures the compressibility of the text, which can indirectly reflect how much redundancy or repetitive patterns are present in the output. Zlib is a widely used compression library that implements the DEFLATE algorithm. This score is computed by:

$$\text{zlib\_score} = \frac{\text{length of compressed output}}{\text{length of original text}}$$

A lower zlib score indicates higher compressibility, meaning the text contains more redundant or repetitive information.

The DEFLATE algorithm, which zlib is based on, was designed by Phil Katz and first introduced in the 1990s [35]. It combines LZ77 compression and Huffman coding to effectively reduce the size of data while preserving the original information. In the context of evaluating text models, the zlib score helps quantify how much the generated text can be compressed, providing another dimension to assess the model's memorization and output characteristics.

B.3 Comparison of PPL & Zlib Scores

B.3.1 Experimental Procedures

Similar to the way the results were analyzed for results of the memorization metric, the results of the different model sizes on the perplexity and Zlib scores were also measured, to see if there were any trends between them. For this, the model output was analyzed by the tokenizer and model and the zlib evaluated the model's output string.

B.3.2 Effect on Model Sizes

The results for the Zlib score indicate that the Mamba and the Pythia have extremely similar results, with the results looking like they overlap. The average zlib score appears to remain relatively the same throughout changes of the model's size. However, when evaluating the average perplexity of the models, the Mamba model shows less perplexity than the compared Pythia model. This is notable as some literature notes that perplexity reflects higher memorization, but it was shown that Mamba memorizes more than Pythia.

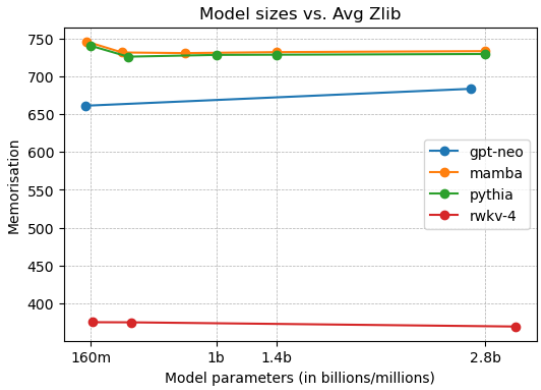


Figure B.1: Average zlib score of the model's output compared to the model sizes.

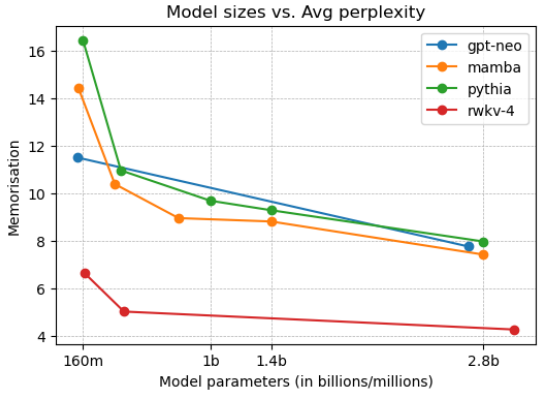


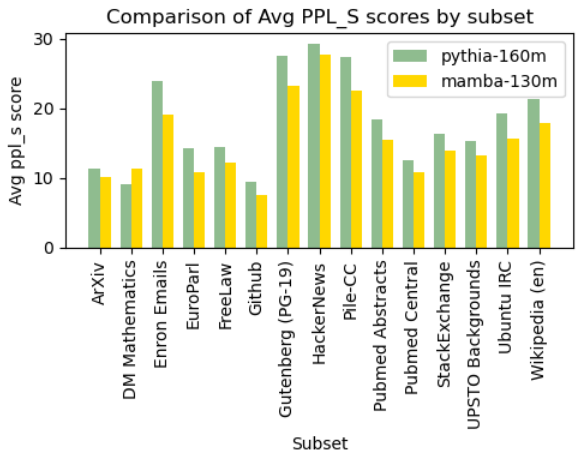
Figure B.2: Increasing model sizes against the average perplexity of the model.

B.3.3 Effect on Data Subsets

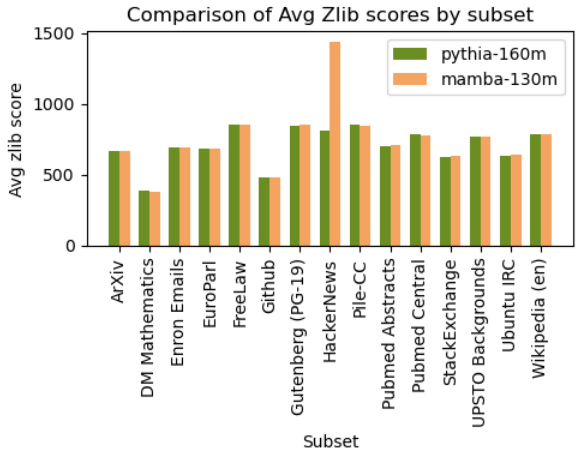
When analyzing the results from the prefix attacks, the Zlib and perplexity metrics revealed that the DM Mathematics subset exhibited lower perplexity and Zlib scores. This suggests a higher level of memorization for both exact and fuzzy memorization. However, this was not reflected in the actual text. In contrast, subsets like EuroParl and PubMed Abstracts also showed lower perplexity, but they exhibited minimal memorization in both fuzzy and exact metrics. This indicates that perplexity scores might be more indicative of memorization in certain domains or when comparing specific samples across different subsets.

B.4 Min-K% and Min-K%++ Comparison

A comparison of the line plots confirmed the findings of the Min-K%++ research [71], with the Min-K%++ method showing less deviation for different values of k. It is also



(a) PPL



(b) Zlib

Figure B.3: Results of small Mamba and Pythia models for membership inference metrics on subsets of The Pile: (a) perplexity (PPL) and (b) Zlib.

visible that the AUROC values are higher for the Min-K%++ experiment. It is important to note that the k values may aid in identifying patterns to unlearn for the model. While the Min-K% method has been proven to help with defense, the Min-K%++ method has not yet been used in that manner. It is also interesting to note that both models exhibit similar responses and patterns, although the Mamba model scores higher in the Min-K% evaluation. Additionally, for potential unlearning tasks, higher k values indicating greater memorization could suggest that the Mamba model might show better performance after initial stages of unlearning, given its propensity to memorize more during training.

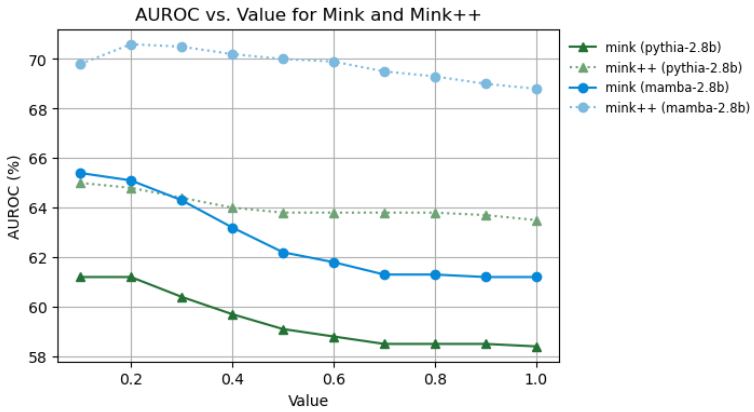


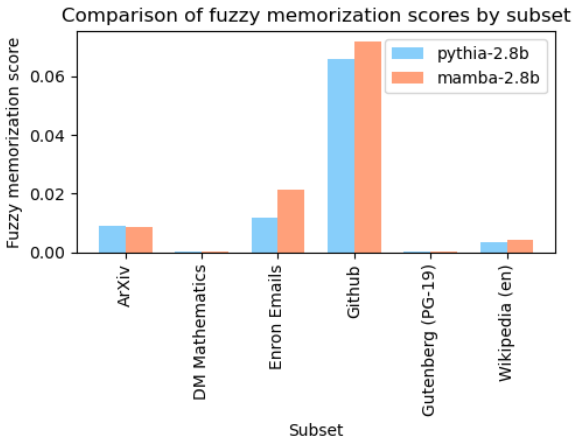
Figure B.4: Comparison of the Min-K% PROB, represented by mink and Min-K%++, represented by mink++, methods across different values of k for 2.8b sized Mamba and Pythia models. The figure illustrates how the performance metrics, such as AUROC, vary with changes in k. Min-K%++ generally shows less deviation and higher AUROC scores compared to Min-K%.

Appendix C

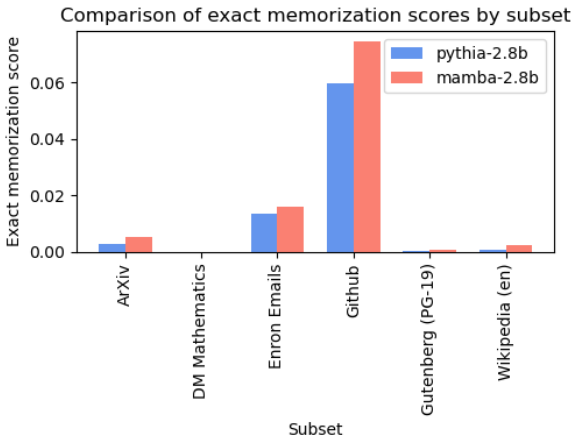
Additional Prefix Attack

C.1 Subset Analysis on Large Models

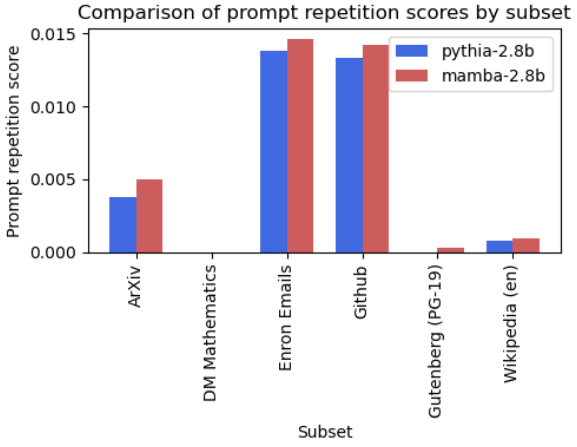
The results for the larger run models on the subset data indicate similar patterns of memorization for the subsets. However, the prompt repetition measured for the larger subsets when compared to the smaller experiment run Figure 3.6c that the Mamba models show greater repetition than Pythia models for larger sized models in Figure C.1c. Additionally, the prompt repetition for the Enron Emails subset becomes slightly larger than that for Github in both models. Interestingly, on the datasets tested, Mamba shows similar patterns of greater memorization for both fuzzy and exact memorization metrics as seen in Figure C.1a and Figure C.1b.



(a) Fuzzy memorization



(b) Exact memorization



(c) Prompt repetition

Figure C.1: Results on 2.8b sized Mamba and Pythia models for memorization metrics: (a) fuzzy memorization, (b) exact memorization, and (c) prompt repetition.

Appendix D

Performance

D.1 Comparison Baselines

While the primary focus of this study was on the Pythia and Mamba models, it is important to contextualize their performance within the broader landscape of models evaluated on this benchmark. Notably, many top-scoring and leaderboard models are significantly larger and fine-tuned beyond the 2.8B parameter size of the Pythia and Mamba models. Therefore, it is crucial to compare these results with those of lower-scoring models to provide a complete picture.

The low scores (0) observed for both the Data Analysis and Coding metrics in the Pythia and Mamba models highlight the challenges of this benchmark. The reference models, as shown in Table D.1, also achieved a score of 0 in these categories. This aligns with the benchmark's characterisation of being exceptionally challenging, as previously outlined.

D.1.1 Reference Model Scores

Model	Global Average	Coding	Data Analysis	Math	Language	Reasoning	Instruction Following
llama-2-7b-chat	10.25	0.00	0.00	44.88	6.86	4.78	5.00
qwen1.5-0.5b-chat	5.26	0.00	0.00	21.30	2.88	3.39	4.00

Table D.1: Performance of reference models across different categories.

When evaluating how other models perform in the Math category Table D.1, the results are comparable to those of the Mamba models, which is promising. However, it

is noted that most models scored lower in Instruction Following metrics compared to the reference models. This disparity is expected, as the reference models are fine-tuned, whereas the main models in this study are baseline models. Consequently, the baseline models generally underperform in these sections.

Overall, it is notable that the highest-performing model from our experiments surpasses the lowest-scoring model on the leaderboard, specifically the Qwen-1.5-0.5b-chat model from the LiveBench leaderboard with the average of 5.26 being lower than the best performig transformer model seen in this study which received an average of 6.678 as seen in Table 4.1. This finding is promising and suggests that the adaptations made to the benchmark for the Mamba models and BTLM model were effective, as demonstrated by their improved performance.

D.2 Model Scores by Task

While the large overview was covered in the main portion of the paper, it can be more beneficial to comprehend the performance of the models and gain a further understanding by looking at how they perform on each task, rather than just on the total averaged benchmark for each section. This chapter divides into a section of each of the respective section tasks and the scores for each.

D.2.1 Coding Tasks

For the Mamba and transformer models trained on The Pile and SlimPajama, the overall benchmark averaged to 0 for all of the models Table 4.1. None of the tasks show a deviance from the original analysis as all indicate 0 here as well Table D.2.

D.2.2 Data Analysis Tasks

For the data analysis tasks it is interesting to note that despite the average of 0 for all of the averaged models on the data analysis benchmark Table 4.1, not all of the results are zero when looking at the individual tasks. The BTLM transformer model was able to retrieve a non-zero score for one of the tasks.

In the TableJoin task, the model is provided with two tables that have partially overlapping columns and is required to create a valid join mapping between them. This task is conceptually similar to the Summarize task, where the model must infer and consolidate information from the provided data. Transformer models have demonstrated

Model	LCB Generation	Coding Completion
Mamba-130m	0.00	0.00
Pythia-160m	0.00	0.00
Mamba-370m	0.00	0.00
Pythia-410m	0.00	0.00
Mamba-790m	0.00	0.00
Pythia-1b	0.00	0.00
Mamba-1.4b	0.00	0.00
Pythia-1.4b	0.00	0.00
Mamba-2.8b	0.00	0.00
Pythia-2.8b	0.00	0.00
Mamba-2.8b-SP	0.00	0.00
BTLM-3b-SP	0.00	0.00

Table D.2: Individual scores for each task in the Coding benchmark from LiveBench.

strong performance in the Summarize task, as shown in Table D.4. This suggests that transformer models may also perform well in the TableJoin task due to their ability to effectively handle and integrate complex information.

D.2.3 Instruction Following Tasks

Looking at each individual task for instruction following, for the models Pythia and Mamba that were trained on The Pile, it is interesting to note that while the models averaged to show that the Pythia models showed higher scores for instruction following Table 4.1 that the Mamba model is often better on the story generation metric Table D.4. The prompt of this task is that given a set of sentences, the model is asked ”Please generate a story based on the sentences provided” [66]. When compared to the other tasks that are based on understanding the text, the Pythia transformer models show higher results. However, for results that require less memorization of the prompt, the Mamba demonstrates stronger performance.

Model	CTA	Table Join	Table Reformat
Mamba-130m	0.00	0.00	0.00
Pythia-160m	0.00	0.00	0.00
Mamba-370m	0.00	0.00	0.00
Pythia-410m	0.00	0.00	0.00
Mamba-790m	0.00	0.00	0.00
Pythia-1b	0.00	0.00	0.00
Mamba-1.4b	0.00	0.00	0.00
Pythia-1.4b	0.00	0.00	0.00
Mamba-2.8b	0.00	0.00	0.00
Pythia-2.8b	0.00	0.00	0.00
Mamba-2.8b-SP	0.00	0.00	0.00
BTLM-3b-SP	0.00	**0.58**	0.00

Table D.3: Individual scores for each task in the Data Analysis benchmark from LiveBench.

D.2.4 Language Tasks

For the language tasts, the Pythia models consistently outperform the Mamba models. It is also intresting to note that the Mamba model values seem to increase for sizes 790m and 1.4b, which are when the comparable sized Pythia models decrease in their own performance. This trend then changed for the 2.8b sized model for the models, where the Pythia performs approximately four times as well than the Mamba model.

An interesting observation when looking at the 2.8b slimpajama models is that the mamaba model appears to outperform the transformer model on this metric. Similarly, unlike the Pythia 2.8b sized model that was also able to score on the connections task, the BTLM model is unable to score. This may suggest that the type of data the model is trained on is important, and that the possibility of a de-duplified dataset may result in a more difficult ability to perform well on language tasks.

D.2.5 Math Tasks

In the main metrics, the Mamba models were able to significantly outperform the transformer models for The Pile, but were quite similar on the SlimPajama dataset Table 4.1.

Model	Paraphrase	Simplify	Story Generation	Summarize
Mamba-130m	10.78	13.87	**13.92**	**12.70**
Pythia-160m	**13.02**	**14.53**	11.33	9.70
Mamba-370m	**12.62**	**11.40**	**16.08**	**14.98**
Pythia-410m	11.45	11.20	12.75	12.78
Mamba-790m	12.70	**10.87**	**17.25**	10.83
Pythia-1b	**13.78**	9.20	14.25	**15.75**
Mamba-1.4b	13.70	14.47	**18.42**	17.70
Pythia-1.4b	**23.83**	**15.12**	13.67	**21.12**
Mamba-2.8b	**6.50**	8.17	**12.08**	12.78
Pythia-2.8b	2.33	**16.33**	7.83	**29.40**
Mamba-2.8b-SP	12.53	11.87	9.17	8.45
BTLM-3b-SP	**21.08**	**24.72**	**21.92**	**32.65**

Table D.4: Individual scores for each task in the Instruction Following benchmark from LiveBench. Models that are bolded indicate the highest overall performance within their model size and dataset grouping, while bolded numbers represent the highest result for each task within the respective grouping.

It is interesting to note that within the individual metrics that for the Mamba 2.8b model was a slight reduction in performance on the Olympiad benchmark, but was able to score on the AMPS Hard metric. The Olympiad benchmark is based in questions from international IMO competitions that are prestigious competitions for high school students. Additionally, the AMPS dataset is are synthetically generated math questions that are harder as random primitives are drawn from a larger and more difficult distribution across the 10 most difficult tasks within AMPS. For this metric, it is notable that a non-tuned Mamba model was able to score on this metric.

Another thing to notice from the table is that the SlimPajama models were not able to score as well on these metrics. This demonstrates the importance of data within the models training and demonstrate that the distribution of data within the dataset could be important in the baseline models ability to perform.

Model	Connections	Plot Unscrambling	Typos
Mamba-130m	0.00	**5.02**	0.00
Pythia-160m	0.00	4.87	0.00
Mamba-370m	0.00	4.40	0.00
Pythia-410m	0.00	**6.94**	0.00
Mamba-790m	0.00	6.08	0.00
Pythia-1b	0.00	**6.36**	0.00
Mamba-1.4b	0.00	6.04	0.00
Pythia-1.4b	0.00	**7.36**	0.00
Mamba-2.8b	0.00	**4.37**	0.00
Pythia-2.8b	0.00	4.12	0.00
Mamba-2.8b-SP	0.00	**7.13**	0.00
BTLM-3b	0.00	5.64	0.00

Table D.5: Individual scores for each task in the Language benchmark from LiveBench. Models that are bolded indicate the highest overall performance within their model size and dataset grouping, while bolded numbers represent the highest result for each task within the respective grouping.

D.2.6 Reasoning Tasks

Within the Mamba and Pythia models in the averaged category of reasoning Table 4.1 the Mamba model proved to have higher overall performance on the reasoning task for the 370m when compared to Pythia sized 410m. Notably, when looking at the task breakdown for the reasoning benchmark in Table D.7, the results show that for the models of that size, each only scored non-zero for different tasks. The Web of Lies v2, is a harder version of Big-Bench Hard [58] that evaluates the truth for a boolean function that is expressed as a word problem. The task Zebra Puzzles was based off of the existing well-known reasoning task [32] that tests models on their ability logically deduce information given a list of statements that create constraints. Additionally, the Mamba models are unable to score on the Web of Lies v2 task, but are able to score on the Zebra Puzzle task.

This suggests that the Mamba model may be capable of reasoning through logic, but may struggle with following instructions in language. This is supported by the

Model	AMPS Hard	Olympiad	Math Comp
Mamba-130m	0.00	**12.43**	0.00
Pythia-160m	0.00	11.99	0.00
Mamba-370m	0.00	**9.58**	**1.04**
Pythia-410m	0.00	5.28	0.00
Mamba-790m	0.00	**6.92**	0.00
Pythia-1b	0.00	0.00	0.00
Mamba-1.4b	0.00	**13.36**	0.00
Pythia-1.4b	0.00	6.35	0.00
Mamba-2.8b	1.00	**9.57**	0.00
Pythia-2.8b	0.00	0.23	0.00
Mamba-2.8b-SP	0.00	1.31	0.00
BTLM-3b-SP	0.00	**2.70**	0.00

Table D.6: Individual scores for each task in the Math benchmark from LiveBench. Models that are bolded indicate the highest overall performance within their model size and dataset grouping, while bolded numbers represent the highest result for each task within the respective grouping.

higher scores of the Mamba model on the mathematical benchmarks that have a greater focus on logic Table D.6 and the Pythia models performing higher on the instruction following tasks Table D.4.

Model	Web of Lies v2	Zebra Puzzle
Mamba-130m	0.00	2.00
Pythia-160m	0.00	2.00
Mamba-370m	0.00	**10.00**
Pythia-410m	**2.00**	0.00
Mamba-790m	0.00	**8.00**
Pythia-1b	**4.00**	4.00
Mamba-1.4b	0.00	**2.00**
Pythia-1.4b	**8.00**	0.00
Mamba-2.8b	0.00	0.00
Pythia-2.8b	**2.00**	**4.00**
Mamba-2.8b-SP	0.00	0.00
BTLM-3b-SP	**4.00**	**20.00**

Table D.7: Individual scores for each task in the Reasoning benchmark from LiveBench. Models that are bolded indicate the highest overall performance within their model size and dataset grouping, while bolded numbers represent the highest result for each task within the respective grouping.