Wrestling Mamba: Exploring Early Fine-Tuning Dynamics on Mamba and Transformer Architectures

Stanford CS224N Custom Project

Lucas Brennan-Almaraz  
Department of Computer Science  
Stanford University  
leba@stanford.edu

Daniel Guo  
Department of Computer Science  
Stanford University  
danguo@stanford.edu

Sarvesh Babu  
Department of Computer Science  
Stanford University  
sarveshb@stanford.edu

Abstract

The Mamba architecture for large language models has recently emerged as a promising alternative to widely-used Transformer-based architectures. However, due to its novelty, there is a lack of literature and open-source tools available for evaluating and comparing the performance of Mamba models. In this paper, we address this gap by pursuing two main objectives: (1) enhancing open-source infrastructure for fine-tuning Mamba models in a well-documented manner, and (2) conducting a rigorous benchmark study to investigate the fine-tuning stability of Mamba models in comparison to Transformer-based models. To achieve our first objective, we fixed the existing bugs in the HuggingFace Transformers Library and provided comprehensive process documentation. For our second objective, we performed benchmark comparisons for fine-tuning stability using a question-answering dataset SlimOrca. In this preliminary study, we investigated the effects of fine-tuning the Mamba model using LoRa for a single epoch and compared its performance to other transformer architectures. Although our experimental results were mixed, we made a significant contribution by developing code that enables, to the best of our knowledge, the first successful integration of LoRa fine-tuning with the Mamba architecture.

1 Key Information

- Mentor: Tianyi Zhang
- External Collaborators: Sarvesh Babu (Stanford Undergraduate)
- Sharing project: N/A
- Team Contributions: Lucas: analyze fine-tuning stability, Daniel: compare effects of fine-tuning across Transformer models and Mamba, Sarvesh: investigate modifications to HF Transformers Library

2 Introduction

Large language models (LLMs) have emerged as a keystone development in the field of artificial intelligence by demonstrating remarkable capabilities in understanding, generating, and manipulating human language. The evolution of LLMs, from early neural network-based models to the current state-of-the-art Transformer architectures, has been driven by the increasing scale of high-quality
massive datasets and compute. However, little has changed architecturally since 2017, when Transformers with Attention \cite{Vaswani2017} became the dominant architecture for LLMs \cite{Brown2020}.

There are two main drawbacks to this current paradigm. Firstly, massive scaling of attention-based Transformer models enhances performance but also increases computational complexity and costs due to the self-attention mechanism, which grows quadratically with sequence length \cite{Vaswani2017}. At their current scales, the training of state-of-the-art LLMs is extremely expensive and requires abundant resources, which raises concerns regarding equitable access and downstream environmental impacts. Secondly, the fixed-size context window of attention-based Transformer models limits their ability to capture long-range dependencies and maintain coherence over extended sequences \cite{Beltagy2020}. This constraint hinders their application in tasks that require understanding and generating longer and more complex sequence data.

The Mamba architecture introduced by \cite{Gu2023} addresses these problems by leveraging a selective state-space model that instead has linear scaling computational complexity and memory, which solves the first problem. The second problem of limited context length is solved by the state space formulation which is meant for continuous sequences, which theoretically allows for much longer sequences when input sequences are discretized.

Our aim was to produce benchmarks comparing fine-tuning stability between Mamba and Transformer-based models to contribute to existing literature and improve upon existing open-source tools that work with Mamba as that was a pain point early in our research. Our work produced direct edits to the HuggingFace (HF) Transformers Library that provided support for Mamba quantization and benchmarks that demonstrated it is possible to finetune Mamba with LoRA.

3 Related Work

3.1 Performance Comparisons between Transformers and Mamba

In \cite{Gu2023}, the authors compare the vanilla Mamba model (pre-trained on the Pile) with state-of-the-art Transformer-based models across varying parameter sizes. However, to our knowledge, there have not been any studies that focus solely on comparing performance and training stability when fine-tuning Mamba and Transformer-based models.

3.2 In-Context Learning Ability Comparisons between Transformers and Mamba

Setting aside raw benchmarks for overall performance, a core utility of the Transformer is its ability to perform in-context learning (ICL). \cite{Grazzi2024} exploration of Mamba’s ability to perform ICL compared to Transformers found that Mamba’s strong ICL performance suggests that it can effectively adapt to new tasks using only a few examples, without the need for extensive fine-tuning. This adaptability potentially makes it easier to fine-tune Mamba for Question Answering tasks, as it may require less data and training time than models with weaker ICL abilities. Moreover, \cite{Grazzi2024} observes that Mamba optimizes its internal representations incrementally like Transformers, suggesting that the fine-tuning process for Mamba might resemble that of Transformers. This internal representation change for in-context learning implies that techniques used to fine-tune Transformers potentially be applied to Mamba as well. This paper greatly informed our research design.

4 Approach

We fine-tuned Mamba-2.8b on the SlimOrca-Dedup dataset for our experimental model. We chose this model size because it is the largest and most powerful open-source version that is available on HF. Subsequent studies can apply our results as a benchmark for further research.

4.1 Baselines and Model Choice

To provide a comprehensive and fair comparison with our experimental model, Mamba-2.8b, we selected four state-of-the-art Transformer-based models: Gemma-2b, Llama-2-7b, Pythia-2.8b, and Qwen-1.8b. In each model suite, we chose the size that is closest to Mamba-2.8b. These models
were chosen based on factors such as novelty, model size, and language-modeling performance. For each model, we downloaded the pre-trained models from HF and fine-tuned them from scratch on the SlimOrca-Deduped dataset using our custom hyper-parameters as outlined in Table 2.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Attention</th>
<th>Normalization</th>
<th>Activation Function</th>
<th>Positional Embeddings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gemma-2b</td>
<td>Decoder-only</td>
<td>Multi-Query</td>
<td>RMSNorm</td>
<td>GeGLU</td>
</tr>
<tr>
<td>Llama-2-7b</td>
<td>Decoder-only</td>
<td>Multi-Headed</td>
<td>RMSNorm</td>
<td>SwiGLU</td>
</tr>
<tr>
<td>Pythia-2.8b</td>
<td>Decoder-only</td>
<td>Multi-Headed</td>
<td>LayerNorm</td>
<td>GeLU</td>
</tr>
<tr>
<td>Qwen-1.8b</td>
<td>Decoder-only</td>
<td>Multi-Headed</td>
<td>RMSNorm</td>
<td>SwiGLU</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Baseline Transformer-based Model Architectures

Gemma-2b, released by Google in February 2024, represents the latest advancements in open-sourced industry models. Despite its relatively small size, Gemma has demonstrated remarkable performance, matching or surpassing larger Transformer-based models on various benchmarks. This performance can be attributed to the use of high-quality training data and a larger vocabulary size (Team et al., 2024). By comparing Mamba-2.8b with Gemma-2b, we aim to investigate the impact of these factors on model performance, as Mamba was trained on the Pile dataset without the benefit of these enhancements.

Llama-2-7b, released by Meta and Microsoft in July 2023, serves as our "gold standard" benchmark. As a larger open-source model, Llama-2-7b sets a high bar for performance (Touvron et al., 2023). We chose this larger model because Mamba can perform comparatively to Transformer models twice its size, as outlined in Gu and Dao (2023). If Mamba-2.8b, with its novel architecture, can surpass the performance of Llama-2-7b on our evaluation tasks, it would be a significant achievement.

Pythia-2.8b was included as a representative of GPT3-like models, providing a baseline comparison against recent state-of-the-art models without special data curation techniques (Biderman et al., 2023). Notably, Pythia-2.8b was trained on the same dataset as Mamba-2.8b—the Pile—allowing for a direct comparison of the Mamba architecture against a Transformer-based model when pre-trained on identical data.

Qwen-1.8b was included due to its unique property of closely following scaling laws as computational resources are increased. Qwen has been shown to consistently improve performance in accordance with a well-defined function as computational power is scaled up (Bai et al., 2023). This characteristic makes Qwen-1.8b an ideal stable comparator for evaluating the performance of Mamba-2.8b across different training runs and computational resource allocations.

4.2 Structured State-Space Architecture (S4)

Gu et al. (2022) introduced the Structured State Space (S4) sequence model to address the challenge of efficiently modeling long-range dependencies (LRDs), which Transformers struggle to tackle. S4 is based on a novel parameterization of the state space model (SSM) that allows for faster computation compared to previous approaches while maintaining their theoretical strengths. The state space model is defined simply by the following equations where \( u(t), y(t) \) are scalar input and output respectively, \( x(t) \) is an N-dimensional latent state, and \( A, B, C, D \) are matrices:

\[
x'(t) = Ax(t) + Bu(t) \\
y(t) = Cx(t) + Du(t)
\]

The core idea is to parameterize the state matrices \( A \) by decomposing them as the sum of a low-rank approximation and a normal matrix, enabling efficient computation. By combining this parameterization with a truncated generating function in frequency space and the Woodbury identity, the authors reduced the SSM to a well-known Cauchy kernel, achieving linear computational complexity and memory usage.

S4 demonstrated significant empirical improvements, setting new state-of-the-art results on various LRD tasks, including the Long Range Arena benchmark and raw speech classification, and showcased...
its potential as a general-purpose sequence model in settings such as generative modeling, image classification, and time series forecasting. However, S4 lagged behind Transformers in non-LRD tasks as it lacked the equivalent capability of Attention in Transformers.

### 4.3 Selective State-Space Architecture (Mamba)

Building upon the foundations of S4, Gu and Dao (2023) introduced Mamba, a new architecture that claims to achieve the modeling power of Transformers while maintaining linear time and space complexity scaling with sequence length. They improved upon S4 by introducing a new selection mechanism that enables the model to selectively propagate or forget information along the sequence length dimension based on the current token instead of remaining static. Although this change prevents the use of efficient convolutions, the authors designed a hardware-aware parallel algorithm in recurrent mode to overcome this challenge. The selective SSMs are integrated into a simplified end-to-end neural network architecture that does not rely on attention or MLP blocks.

![Figure 1: Mamba Architecture Diagram from Gu and Dao (2023)](image)

![Figure 2: Pseudocode for S4 vs Mamba from Gu and Dao (2023)](image)

### 4.4 Methods

At a high level, we compared the results of LoRa (low rank approximation) fine-tuning between Mamba and Transformers.

#### 4.4.1 8-bit Quantization of Baseline Models

To reduce the memory footprint and computational cost, we quantize the pre-trained model weights (Transformer models) to 8 bits. Quantization compresses the model by representing weights with lower-precision values, resulting in smaller model sizes and faster computations. We were extremely compute and time-bound therefore this was a necessary sacrifice. We intended to do this for Mamba as well, but due to time constraints, we couldn’t figure out the code changes for the HF Transformers Library to quantize Mamba and there are currently no open-sourced quantization scripts for Mamba. We wish to expand the open-source fine-tuning code base even further from our own contribution.

#### 4.4.2 Low-Rank Adaptation (LoRa)

We adopt the LoRa technique, a parameter-efficient fine-tuning technique that freezes the pre-trained model weights and injects small trainable matrices into each layer of the model (Hu et al., 2021). For
a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRa reparameterizes it as:

$$W_0 + \Delta W = W_0 + B A$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During fine-tuning, $W_0$ is frozen and the trainable parameters are the low-rank matrices $A$ and $B$. This greatly reduces the number of trainable parameters compared to fine-tuning all weights.

4.4.3 Applying LoRa to Mamba

We extend the LoRa approach to the Mamba architecture, which has not been explored in previous literature. By injecting trainable low-rank matrices into the Mamba layers, we aimed to efficiently compress this novel architecture in memory while preserving its computational advantages in language modeling tasks.

4.4.4 Fine-tuning Procedure

For both Transformer and Mamba models, we (1) add LoRa modules to the appropriate layers (e.g., self-attention and feed-forward layers in Transformers; Mamba blocks), (2) freeze the pre-trained model weights, and (3) train only the LoRa parameters along with layer normalization and bias parameters. The LoRa rank $r$ allows for tuning the trade-off between model expressiveness and parameter efficiency. By combining quantization and LoRa, we can efficiently fine-tune large Transformer and Mamba models on downstream tasks under compute and time constraints. Applying LoRa to Mamba is a novel and experimental approach that has not been previously explored. We evaluate our fine-tuned models on language modeling metrics using the EleutherAI Evaluation Harness. After fine-tuning, the trained LoRa weights can optionally be merged back into the frozen pre-trained matrices for deployment without adding inference latency.

5 Experiments

5.1 Data

We used the SlimOrca-Deduped dataset which is a curated and deduplicated subset of the OpenOrca augmented FLAN text data that aligns with the distributions in the Orca paper (Mukherjee et al., 2023). SlimOrca-Deduped contains 363,491 entries (307 MB) of augmented FLAN data used for researchers to achieve GPT-4 quality for fine-tuned models. Specifically, the SlimOrca dataset is used for language modeling tasks such as Q&A, language generation, and reasoning. The structure of SlimOrca is in ShareGPT using the ChatML template.

5.2 Evaluation method

We used the EleutherAI evaluation harness to comprehensively test our fine-tuned models across several accredited benchmarks and natural language tasks. The specific evaluation methods that we ran were LAMBADA (OpenAI), HellaSwag, PIQA, ARC Easy, ARC Challenge, and WinoGrande, which focus on performance related to reasoning and question answering. We chose these specific methods since they are used in the original Mamba paper and we wanted to compare our results with existing literature.

5.3 Experimental details

We trained our baseline Transformer models using our own custom fine-tuning scripts that make use of the HF Transformers Library. Each fine-tuning run took roughly 45-60 hours to complete on a single A100 GPU. We fine-tuned all the model in parallel on separate Runpod instances. We defined our hyperparameters for each model as detailed in Table 2.

5.3.1 Supervised Fine-tuning with LoRa Implementation

To accomplish our finetuning, we first attempted to use the open source Axolotl fine-tuning tool for all of our models. However, it became apparent there were bugs with Mamba compatibility. To maintain consistent hyperparameters and methods, we reverted to writing our own fine-tuning scripts using a
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Batch Size</td>
<td>2</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.0002</td>
</tr>
<tr>
<td>Gradient Accumulation Steps</td>
<td>4</td>
</tr>
<tr>
<td>Dropout Rate</td>
<td>0</td>
</tr>
<tr>
<td>Warmup Steps</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Sequence Length</td>
<td>2048</td>
</tr>
<tr>
<td>Optimizer</td>
<td>AdamW</td>
</tr>
<tr>
<td>Lora Rank</td>
<td>16</td>
</tr>
<tr>
<td>Lora Dropout</td>
<td>0</td>
</tr>
<tr>
<td>Lora Bias</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 2: Supervised Fine-Tuning Hyperparameters

modified version of the HF Transformers Library. Our final script for fine-tuning the Transformer models incorporated both LoRA adapters for learning and 8-bit quantization of encodings of the pre-trained models.

5.3.2 Infrastructure for Fine-tuning Mamba

To address the lack of open-source tools for fine-tuning Mamba models, we first focused on developing and enhancing the existing infrastructure. We identified and fixed bugs in the available open-source fine-tuning code and created comprehensive documentation to facilitate the fine-tuning process for Mamba models. Specifically, we aimed to make Mamba interoperable with the HF Transformers library infrastructure for fine-tuning.

On March 9, 2024, the HF Transformers library began to add interoperability for Mamba with their infrastructure, and we updated their infrastructure to work properly. The main bug that was our contribution to the open source implementation was (1) finding the correct dependency installation that is stable and (2) editing the Transformers library to fix the issue regarding the bias terms in the Mamba layers when wrapping the Transformers library around the Mamba architecture. We open-sourced our repository, and after further validation and testing of the stability we aim to make a pull request to the HuggingFace Library.

5.3.3 Experiment 1: Language Modelling Performance Comparison

In the first experiment, we compared the performance of the fine-tuned models on language modeling and question-answering tasks. We employed the Eleuther AI test harness to evaluate the models’ performance on our five chosen benchmarks: LAMBADA (OpenAI), HellaSwag, PIQA, ARC easy, ARC Challenge, and WinoGrande.

5.3.4 Experiment 2: Fine Tuning Stability/performance change experiment

For the second experiment, we investigated the model improvement over time during the early training stage (1 epoch). We used the EleutherAI test harness to assess the fine-tuning stability across the Mamba and Gemma architectures when trained at a similar rate (we only compared one transformer due to time and compute constraints). This allowed us to perform preliminary comparisons between the fine-tuning characteristics of the two architectures.

5.4 Results

5.4.1 Experiment 1: Language Modelling Performance Comparison

The fine-tuning performance results for Experiment 1 are showcased in Fig. 3. We map the ARC-E, ARC-C, HellaSwag, LAMBADA, piqa and wino-grande accuracies as well as the average accuracy and LAMBADA perplexity. We compare the numbers across each vanilla model, quantized Transformer models, and Fine-tuned models.
Table 3: Performance metrics from EleutherAI test harness of all models. Bolded models are unquantized and vanilla versions.

<table>
<thead>
<tr>
<th>Model</th>
<th>ARC-E acc</th>
<th>ARC-C acc</th>
<th>HellaSwag acc</th>
<th>LAMBADA acc</th>
<th>piqa acc</th>
<th>wino-grande acc</th>
<th>Average acc</th>
<th>LAMBADA perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gemma-2b</td>
<td>0.7412</td>
<td>0.3993</td>
<td>0.5260</td>
<td>0.6227</td>
<td>0.7677</td>
<td>0.6527</td>
<td>0.6183</td>
<td>5.5445</td>
</tr>
<tr>
<td>Quantized</td>
<td>0.7420</td>
<td>0.4019</td>
<td>0.5265</td>
<td>0.6410</td>
<td>0.7677</td>
<td>0.6519</td>
<td>0.6218</td>
<td>5.1974</td>
</tr>
<tr>
<td>Quantized + Fine-tuned</td>
<td>0.6111</td>
<td>0.3447</td>
<td>0.4990</td>
<td>0.5991</td>
<td>0.7573</td>
<td>0.6582</td>
<td>0.5782</td>
<td>5.4337</td>
</tr>
<tr>
<td>Llama-2.7b</td>
<td>0.7597</td>
<td>0.4386</td>
<td>0.5733</td>
<td>0.7403</td>
<td>0.7786</td>
<td>0.6882</td>
<td>0.6631</td>
<td>3.4048</td>
</tr>
<tr>
<td>Quantized</td>
<td>0.7386</td>
<td>0.4411</td>
<td>0.5778</td>
<td>0.7099</td>
<td>0.7639</td>
<td>0.6638</td>
<td>0.6492</td>
<td>3.2625</td>
</tr>
<tr>
<td>Quantized + Fine-tuned</td>
<td>0.7054</td>
<td>0.4189</td>
<td>0.5654</td>
<td>0.6916</td>
<td>0.7353</td>
<td>0.6756</td>
<td>0.6351</td>
<td>3.5567</td>
</tr>
<tr>
<td>Qwen-1.8b</td>
<td>0.6511</td>
<td>0.3140</td>
<td>0.4545</td>
<td>0.5785</td>
<td>0.7280</td>
<td>0.6140</td>
<td>0.5568</td>
<td>7.4802</td>
</tr>
<tr>
<td>Quantized</td>
<td>0.6473</td>
<td>0.3217</td>
<td>0.4540</td>
<td>0.5802</td>
<td>0.7339</td>
<td>0.6117</td>
<td>0.5581</td>
<td>7.224</td>
</tr>
<tr>
<td>Quantized + Fine-tuned</td>
<td>0.6296</td>
<td>0.3225</td>
<td>0.4565</td>
<td>0.5703</td>
<td>0.7301</td>
<td>0.6117</td>
<td>0.5535</td>
<td>7.4547</td>
</tr>
<tr>
<td>Pythia-2.7b</td>
<td>0.6423</td>
<td>0.2952</td>
<td>0.4526</td>
<td>0.6433</td>
<td>0.7443</td>
<td>0.5848</td>
<td>0.5604</td>
<td>12.424</td>
</tr>
<tr>
<td>Quantized</td>
<td>0.6418</td>
<td>0.2952</td>
<td>0.4533</td>
<td>0.6464</td>
<td>0.7399</td>
<td>0.5951</td>
<td>0.4657</td>
<td>5.0396</td>
</tr>
<tr>
<td>Quantized + Fine-tuned</td>
<td>0.5821</td>
<td>0.2824</td>
<td>0.4389</td>
<td>0.6352</td>
<td>0.7301</td>
<td>0.6125</td>
<td>0.5469</td>
<td>5.2282</td>
</tr>
<tr>
<td>Mamba-2.8b</td>
<td>0.6970</td>
<td>0.3430</td>
<td>0.4951</td>
<td>0.6907</td>
<td>0.7508</td>
<td>0.6338</td>
<td>0.6017</td>
<td>4.2308</td>
</tr>
<tr>
<td>Fine-tuned</td>
<td>0.5602</td>
<td>0.2901</td>
<td>0.4874</td>
<td>0.6899</td>
<td>0.7557</td>
<td>0.6377</td>
<td>0.5701</td>
<td>4.2117</td>
</tr>
</tbody>
</table>

5.4.2 Experiment 2: Fine-tuning Stability

The performance comparisons between Gemma-2b and Mamba-2.8 during fine-tuning are showcased below.

Figure 3: Gemma-2b Results

Figure 4: Mamba-2.8B Results

6 Analysis

Fine-tuning instability is a well-known challenge in adapting pre-trained language models to downstream tasks, particularly when working with limited training resources (Dodge et al., 2020). In this study, we observe a clear trend of performance degradation after just one epoch of LoRa fine-tuning across all tested Transformer models, aligning with expectations set by prior work. Our primary interest was to investigate whether the Mamba model exhibited different fine-tuning dynamics compared to the other architectures.

Upon analyzing the results, we notice that Mamba follows the same pattern of instability as the other Transformer models. The Vanilla Mamba model outperforms its fine-tuned counterpart across almost all evaluation metrics except for piqa, suggesting that a single epoch of adaptation is insufficient to yield improvements.

We hypothesize that the fine-tuning instability observed in this study can be attributed to insufficient training. With only one epoch, the models are prone to overfitting to noise in the fine-tuning data and failing to converge to optimal solutions. This behavior is expected given the nature of gradient descent optimization, which typically requires multiple passes over the training set to reach stable
convergence (Ruder 2017). However, Mamba’s stable performance on the Question and Answer task benchmark - Piqa - is interesting and warrants further exploration.

Even then, this stability in Piqa is likely caused by a larger batch size when training Mamba to fit the time constraint of the class. This is as batch size determines the number of examples used to compute the gradient at each training step. Smaller batch sizes lead to noisier gradient estimates because they are based on a limited subset of the data. This noise can cause the model to take less stable update steps, leading to potential oscillations or divergence during fine-tuning. On the other hand, larger batch sizes provide more stable gradient estimates, as they average over a larger number of examples, reducing the impact of individual noisy samples. The results from experiment 2 corrobate this batch size stability hypothesis.

7 Conclusion

In this study, we investigate the fine-tuning stability of the Mamba model compared to other transformer architectures when adapted using LoRa Hu et al. (2021) for a single epoch. Our preliminary results suggest that Mamba could exhibit more stable performance under these conditions specifically for the question answering task and more rapid deterioration of performance for the other tasks. However, we acknowledge that the stable results for the question answering benchmark observed in our first and second experiments may be attributed to the larger batch size used when fine-tuning Mamba - or just randomness that occurs during fine-tuning instability. This choice of a larger batch size was made to obtain preliminary results within the time constraints of the project, which also limited our fine-tuning to a single epoch, as the process took approximately 3-4 days for each run.

While our findings serve as a strong motivation for further experimentation and analysis, we recognize the need for more comprehensive experiments to validate the stability of Mamba under various fine-tuning settings. To facilitate this line of research, we have developed novel modifications to existing open-source libraries, enabling, to our knowledge, the first seamless integration of LoRa fine-tuning with the Mamba architecture.

These technical contributions not only support our current study but also provide a foundation for future investigations into the fine-tuning dynamics of Mamba and other transformer models. By open-sourcing our code and sharing our methodology, we aim to foster collaboration and accelerate progress in the field of language model adaptation.

To build upon this work, we plan to conduct a comprehensive set of experiments involving multiple fine-tuning runs across diverse datasets and tasks. By analyzing the stability of Mamba and other transformer models under various hyperparameter settings and training durations, we seek to gain a deeper understanding of the factors influencing fine-tuning performance and robustness.

Furthermore, we intend to explore the impact of batch size on fine-tuning stability by conducting experiments with consistent batch sizes across all models. This will allow us to isolate the effect of the model architecture on stability and provide a more accurate comparison between Mamba and other transformer models.
References


Riccardo Grazzi, Julien Siems, Simon Schrodi, Thomas Brox, and Frank Hutter. 2024. Is mamba capable of in-context learning?


