

# SOAP and Sonnets: Improving the Optimization Efficiency of GPT-2

Stanford CS224N Default Project

**Kenna Zeng**

Department of Computer Science  
Stanford University  
kennaz@stanford.edu

## Abstract

Increasing the efficiency of training a large language model remains a critical challenge. We want our model to converge quickly in terms of step count as well as wall-clock time. AdamW has become the de facto optimizer for both pretraining and fine-tuning; however, it is still a first-order optimization method. It exhibits instability at higher learning rates, and does not utilize second order information for faster convergence. Recent work introduced SOAP (Shampoo with Adam in the Preconditioner's eigenbasis)[1]. It showed promising results for pretraining by combining Shampoo's preconditioning with Adam's adaptive learning rates. However, its effectiveness for fine-tuning models on downstream tasks remains unexplored.

In this work, comprehensive empirical experiments are conducted to evaluate SOAP for fine-tuning GPT-2 on the following tasks: Paraphrase Detection and Sonnet Generation. A basic implementation to establish baseline performance against AdamW was done. Then, hyperparameter sweeps were conducted across learning rates, weight decay, learning rate scheduling, and preconditioning frequency to maximize final performance. We show the practical advantages of SOAP, namely its faster convergence and stability with higher learning rates, and show that it is a viable alternative to more commonly used first order methods.

## 1 Key Information to include

- Mentor: Shicheng Liu
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any):

| Team member                      | Late days remaining |
|----------------------------------|---------------------|
| Kenna Zeng                       | 6                   |
| Total pooled                     | 6                   |
| Team size                        | 1                   |
| Late days applied for the report | 3                   |

## 2 Introduction

Large pre-trained language models have shown great success in performing downstream tasks through fine-tuning. Improvements in model performance are often achieved through model scaling, architectural changes, or prompt design. The optimizer is often treated as a fixed component, but this is a limiting assumption. By changing the optimization technique, we can potentially improve

convergence speed, stability, and final generalization. This paper aims to explore one of the more recent techniques, SOAP, for fine-tuning two distinct tasks. Our core question is: Does SOAP provide practical gains over standard first-order optimizers (namely AdamW) for GPT-2 fine-tuning? We first compare SOAP and AdamW with common baselines under controlled conditions, tracking endpoint metrics and training behavior. Then, we perform a hyperparameter sweep to find the best learning rates, weight decay, and preconditioning frequency to maximize our accuracy gains.

We make the following contributions:

- Systematic comparison of SOAP vs AdamW on classification and generation fine-tuning of GPT-2.
- We analyze the effects of different hyperparameters on training behavior of SOAP to find out which ones have a larger impact on performance. We show that SOAP is still sensitive to hyperparameter choices, but is stabler when used with a higher learning rate.
- We demonstrate practical advantages of using SOAP, namely quicker convergence and higher stability at larger learning rates.

Our results reinforce the point that optimizer selection does matter and is a meaningful design decision in LLM adaptation. SOAP is a promising option for those who want reliable and efficient convergence. Our findings motivate deeper investigation and thought into how second-order-aware optimizers could be used to our advantage.

### 3 Related Work

In practice, most LLM fine-tuning pipelines default to first-order adaptive optimizers. Adam provided adaptive learning rates for individual parameters based on first and second moments [2]. It was followed by AdamW, which correctly decouples weight decay during the gradient update step[3]. While first-order methods are popular for their low memory cost and simplicity, they ignore the curvature of the loss landscape. They treat every parameter as independent and can also be very sensitive to the learning rate.. This is where second-order methods can help. Traditional methods such as Newton’s Method or L-BFGS use the Hessian to take more direct steps to the minimum. However, computing and inverting the Hessian is impractical due to its high cost. This is where K-FAC and Shampoo introduce the idea of using Kronecker-factorization to make this more manageable [4, 5]. Kronecker-factorization allows us to maintain smaller matrices for each dimension of a tensor. Shampoo can capture inter-parameter correlations with much lower overhead than maintaining the full Hessian.

SOAP improves conditioning with concepts from Shampoo while leveraging the computational efficiency of Adam. Essentially, SOAP runs Adam within the eigenbasis of the preconditioner[1]. Updates are performed directly within this rotated coordinate system, allowing for a more stable system even when the preconditioner is updated infrequently. It is also a simple drop-in replacement for AdamW, while theoretically promising faster convergence.

Recent scaling studies show that architecture and data scale are not the only driver of model quality. Making tweaks in other areas of the pipeline can result in significant improvements. Optimization governs learning efficiency and stability in large transformers [6, 7]. While works involving optimizer evaluations often focus on single-task or pretraining-centric metrics, our work studies the benefits of SOAP across two different downstream objectives.

## 4 Approach

### 4.1 Baseline GPT-2 Model

The base model was implemented using the provided starter code and the standard GPT-2 implementation[8]. For paraphrase detection, the logits for the YES and NO tokens were extracted from the last token, and whichever had the higher value was output as the answer. The baseline results from AdamW are as follows:

- Paraphrase Detection Dev Accuracy: 0.883 after 10 epochs, batch size 256, LR 1e-05
- Sonnet Generation CHRF Score: 41.37 on the 5th epoch, batch size 8, LR 1e-03.

## 4.2 SOAP Implementation

Implementing SOAP involved importing the optimizer code linked in the original paper. It was added as a secondary optimizer option during training. We wanted to see if a simple drop-in replacement of SOAP could achieve comparable performance to AdamW. This goal would be achieved if we were able to get to the TA baselines for each task within 10 epochs. A script to perform a hyperparameter sweep and log metrics at each epoch was also written.

## 5 Experiments

### 5.1 Data

Default Project Datasets: Quora Question Pairs, Sonnet Dataset

### 5.2 Evaluation Method

SOAP was evaluated based on the following metrics:

- Best dev accuracy/CHRF score
- Training Loss
- Wall-Clock time: Total training time and average time per epoch

### 5.3 Experimental Details

#### 5.3.1 Training Details

For paraphrase detection, training was done on Google Colab with a G4 GPU, with a batch size of 256 for both optimizers. AdamW was used with the default learning rate of  $1e^{-5}$ . As one of SOAP's main strengths is its stability, a larger learning rate of  $1e^{-3}$  was used. For sonnet generation, training was done on a H200 GPU with 8 GB RAM within a Modal notebook, with a batch size of 8 for both optimizers. A fixed learning rate of  $1e^{-3}$  was used for both optimizers. Training was capped at 10 epochs.

#### 5.3.2 Hyperparameter sweep

Table 1: Hyperparameter Search Space for SOAP Optimizer on Sonnet Generation

| Hyperparameter              | Values Swept           | Description                           |
|-----------------------------|------------------------|---------------------------------------|
| Learning Rate (LR)          | 1e-4, 1e-3, 2e-3, 5e-3 | Step size for parameter updates       |
| Weight Decay (WD)           | 0.01, 0.05             | L2 regularization coefficient         |
| Precondition Frequency (PF) | 10, 50                 | Steps between preconditioner updates  |
| LR Schedule                 | None, Cosine           | Cosine annealing to 10% of initial LR |

32 configurations in total. Each experiment ran for 10 epochs, and CHRF was evaluated at the final epoch.

Table 2: Hyperparameter Search Space for SOAP Optimizer on Paraphrase Detection

| Hyperparameter     | Values Swept           | Description                     |
|--------------------|------------------------|---------------------------------|
| Learning Rate (LR) | 5e-5, 5e-4, 1e-3, 5e-3 | Step size for parameter updates |
| Batch Size         | 64, 128, 256           | Samples per gradient update     |

12 configurations in total. Each experiment ran for 2 epochs on 10% of the total dataset. Dev accuracy was evaluated after each epoch.

## 5.4 Results

### 5.4.1 Baseline Comparisons

Table 3: Paraphrase Baseline

| Optimizer | Best Dev Accuracy | Final Loss    | Avg. Epoch Time (s) | Total Time (s) | Time to Best (m) |
|-----------|-------------------|---------------|---------------------|----------------|------------------|
| AdamW     | 0.8854            | 0.200         | <b>427.48</b>       | <b>4275.91</b> | 71.2             |
| SOAP      | <b>0.8856</b>     | <b>0.0179</b> | 599.54              | 5995.41        | <b>40.1</b>      |

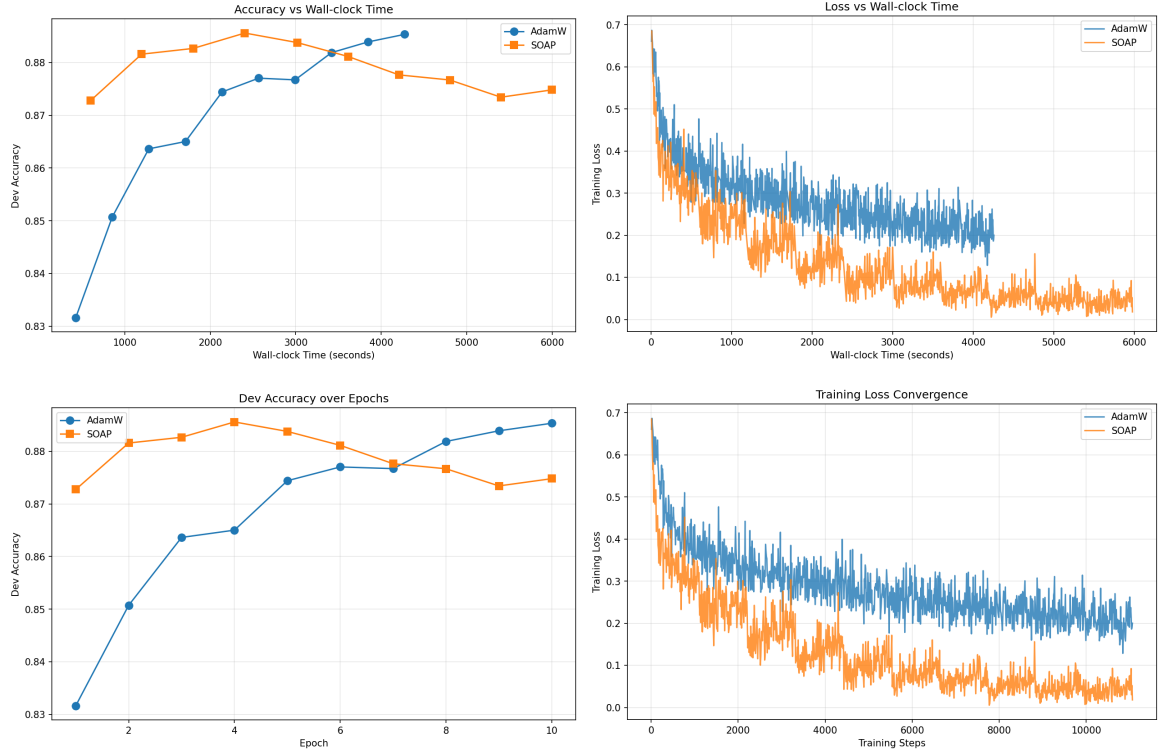
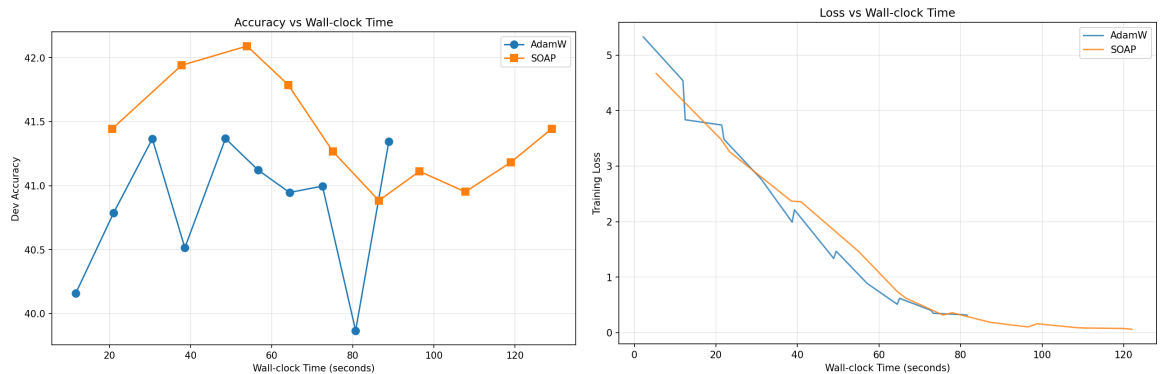
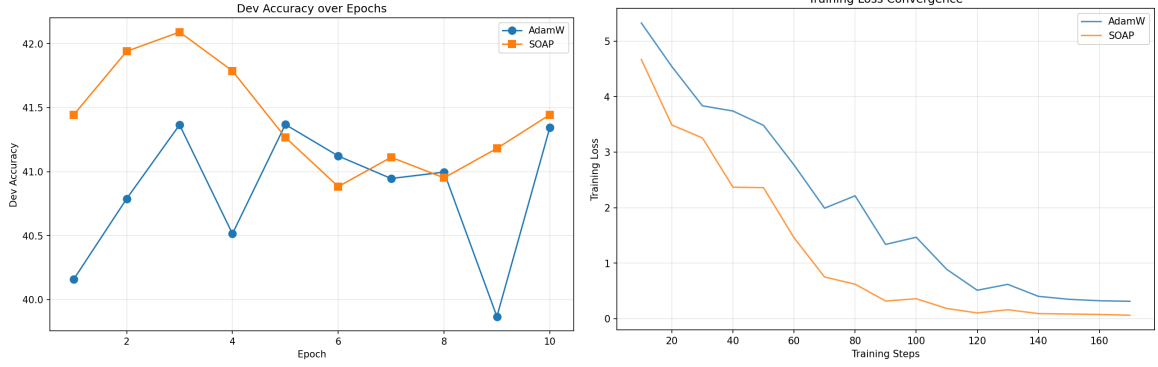


Table 4: Sonnet Baseline

| Optimizer | Best Dev CHRF  | Final Loss    | Avg. Epoch Time (s) | Total Time (s) | Time to Best (s) |
|-----------|----------------|---------------|---------------------|----------------|------------------|
| AdamW     | 41.3681        | 0.3116        | <b>8.89</b>         | <b>88.88</b>   | <b>30.6</b>      |
| SOAP      | <b>42.0907</b> | <b>0.0606</b> | 12.91               | 129.07         | 53.9             |





As a drop-in replacement, SOAP achieved higher best dev accuracy and CHRF score after 10 epochs. These scores were attained at earlier epochs, and final loss was also much lower than AdamW. A single epoch takes about 40% more wall-clock time to complete, but best performance is achieved much earlier with SOAP.

## 5.5 Hyperparameter Sweep

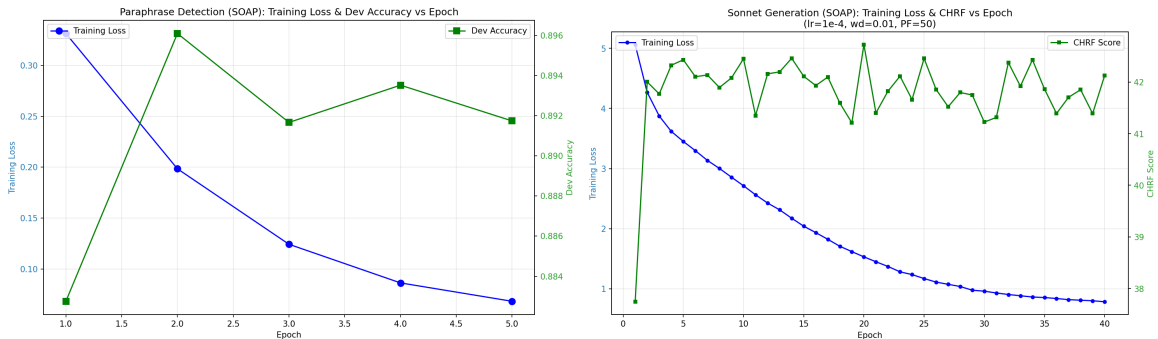
Table 5: Top 5 SOAP Hyperparameter Configurations by Task

| Sonnet Generation |      |      |    |       |    |       | Paraphrase Detection |      |     |       |
|-------------------|------|------|----|-------|----|-------|----------------------|------|-----|-------|
| Rank              | LR   | WD   | PF | Sched | Ep | CHRF  | Rank                 | LR   | BS  | Acc   |
| 1                 | 1e-4 | 0.01 | 50 | Cos   | 9  | 42.37 | 1                    | 5e-4 | 256 | 0.835 |
| 2                 | 1e-4 | 0.05 | 50 | Cos   | 9  | 42.28 | 2                    | 5e-4 | 128 | 0.831 |
| 3                 | 1e-4 | 0.05 | 50 | –     | 9  | 42.23 | 3                    | 5e-4 | 64  | 0.830 |
| 4                 | 1e-3 | 0.05 | 50 | Cos   | 9  | 42.19 | 4                    | 5e-5 | 64  | 0.821 |
| 5                 | 1e-4 | 0.01 | 10 | –     | 9  | 42.10 | 5                    | 1e-3 | 128 | 0.814 |

*Abbreviations:* WD = weight decay, PF = precondition frequency, Sched = LR schedule (Cos = cosine annealing), Ep = best epoch, BS = batch size. Sonnet sweep: 10 epochs, batch size 8. Paraphrase sweep: 2 epochs, 10% data.

## 5.6 Training Results with New Hyperparameters

Both training scripts were run with their top configuration found from the hyperparameter sweep. Paraphrase Detection was run for 5 epochs as peak accuracy was achieved by epoch 2. Sonnet Generation was run for 40 epochs, and reached its highest CHRF score on epoch 20.



## 5.7 Leaderboard Scores

- Paraphrase Test/Dev: 0.877/0.896
- Sonnet Test/Dev: 42.199/42.613

## 6 Analysis

### 6.1 Overall Trends

Across both tasks, SOAP consistently produced better convergence behavior than AdamW, but was prone to overfitting. The main pattern in our results is that SOAP converged to lower loss and reached its best validation performance earlier, while incurring additional overhead per epoch. This is consistent with the intuition behind SOAP: by incorporating curvature information through Shampoo-style preconditioning while retaining Adam-style adaptivity, it can take more effective parameter updates, but those updates are computationally more expensive.

SOAP was not a universally faster optimizer in terms of runtime, but it was a stronger optimizer in terms of convergence quality and stability. For a fine-tuning setting where final performance and robustness matter more than raw throughput, this tradeoff can be worthwhile.

### 6.2 Baseline Comparisons

For paraphrase detection, SOAP slightly outperformed AdamW in best dev accuracy, achieving 0.8856 compared to 0.8854 after 10 total epochs. As expected, SOAP was slower. Its average epoch time was 599.54s compared to 427.48s for AdamW, corresponding to roughly a 40% increase in epoch time. However, SOAP achieved its best validation score on epoch 4 (after 40 minutes), while AdamW’s dev accuracy was still increasing at epoch 10 (after 70 minutes). SOAP’s dev accuracy was higher than AdamW’s during epochs 1–5. This shows faster convergence with SOAP, but it is possible that AdamW’s performance would exceed SOAP’s if more epochs were run. There is a more significant difference in the optimization metrics: SOAP achieved a final loss of 0.0179, compared to 0.2000. The training loss decreased at a faster rate against wall-clock time with SOAP. However, dev accuracy started to drop after epoch 4, even with loss steadily decreasing. The model was overfitting, and early stopping was needed to maintain good performance.

Since paraphrase detection is a classification task with relatively bounded output space, the small gain in accuracy suggests that this task may already be close to saturation under the given GPT-2 setup. In such a regime, improved optimization may not translate into large metric gains, even if the training dynamics are better.

Another interesting observation is that SOAP remained stable despite using a learning rate much larger than AdamW’s. We initially ran SOAP with at AdamW’s default learning rate of  $10^{-5}$ , but with subpar performance. We were able to achieve better performance at  $10^{-3}$ . Achieving a slightly better score under such a larger step size supports the claim that SOAP is more tolerant to aggressive learning rates.

Similar to paraphrase detection, SOAP achieved a higher CHRF score for sonnet generation from the first epoch. SOAP achieved a best dev CHRF score of 42.0907 on epoch 3, improving over AdamW’s 41.3681, which was also attained on epoch 3. The sonnet task is open-ended and generation metrics tend to be noisier, thus the score for both optimizers showed more variance across each epoch. As with paraphrase detection, SOAP also converged to a much lower final loss: 0.0606 compared to 0.3116 for AdamW. There was no significant improvement in loss convergence in terms of wall-clock time, but the loss curve did show better convergence behavior when plotted against step count. SOAP again came with additional overhead and signs of overfitting, but it uses fewer steps to achieve its best performance. Its average epoch time was 12.91s compared to 8.89s for AdamW, an increase of about 45%.

### 6.3 What the Hyperparameter Sweep Shows

The hyperparameter sweeps help clarify SOAP’s performance and which settings mattered most.

For paraphrase detection, the strongest sweep results were all obtained with a learning rate of  $5 \times 10^{-4}$ , regardless of batch size. The top three configurations used batch sizes of 256, 128, and 64, but all shared the same learning rate and achieved very similar accuracies (0.835, 0.831, and 0.830 on the reduced-data sweep). There were slight increases in accuracy as batch sizes increased, which is to be expected. In our setup, learning rate was seen to have a larger effect on SOAP performance than batch size. Extremely small learning rates underused SOAP’s potential, while very large rates such as

$5 \times 10^{-3}$  degraded performance. Thus, SOAP was not insensitive to hyperparameters, but favored using moderate-to-high learning rates.

For sonnet generation, the best configurations were somewhat more distributed, but several trends still emerged. First, the strongest results clustered around learning rates ( $10^{-4}$  to  $1 \times 10^{-3}$ ). In a preliminary sweep, learning rates in the  $10^{-5}$  range were tested, but was outperformed by larger rates. The importance of weight decay is debatable: several of the top configurations used weight decay 0.05, suggesting that stronger regularization was beneficial, however the top configuration used the default weight decay of 0.01. Third, both precondition frequencies 10 and 50 appeared among the top-performing runs, indicating that SOAP retained good performance even when the preconditioner was updated relatively infrequently. This is good because decreasing preconditioner frequency reduces the overhead of SOAP. SOAP can handle a reduction in eigenspace updates without giving up too much optimization quality.

The cosine schedule also appears helpful. Three of the best sonnet configurations used cosine annealing and reached their best CHRF at earlier epochs, which suggests that scheduling can improve training efficiency by allowing SOAP to take larger early steps and refine later. Overall, the sweep results indicate that SOAP benefits from tuning, but the tuning burden is manageable: performance was driven mainly by a small number of factors, especially learning rate.

## 6.4 Takeaways

Together, these results support the argument that SOAP’s updates are more effective, not just larger. In both tasks, SOAP spent more of training in a lower-loss regime than AdamW. This is the kind of behavior we would expect from an optimizer that better accounts for curvature and parameter interactions. Stability is also supported by the fact that SOAP remained effective at larger learning rates. In practice, one of the most frustrating aspects of AdamW fine-tuning is that a learning rate that is slightly too large can quickly destabilize training. SOAP did not remove hyperparameter sensitivity, but our results suggest that it broadened the range of usable learning rates and allowed more aggressive updates without collapse.

Despite these optimization benefits, SOAP did not dominate AdamW on raw performance, and there were efficiency tradeoffs. On both tasks, the per-epoch overhead was substantial. This is unsurprising, since SOAP must maintain and periodically update additional preconditioning statistics. In a small-scale fine-tuning setup such as ours, that overhead can outweigh the runtime benefit of requiring fewer effective epochs. If the goal is instead to obtain stronger convergence, more stable training, or better final generation quality, SOAP is a viable option.

These findings support our broader claim that optimizer choice is not a minor implementation detail. Even without changing the model architecture or dataset, replacing AdamW with SOAP changed the optimization trajectory in measurable and practically relevant ways.

## 6.5 Limitations of the Study

Several limitations should be noted. First, the two tasks were run on different hardware environments, so wall-clock comparisons should only be interpreted within each task and not across tasks. Second, the paraphrase sweep was performed on only 10% of the data for two epochs, which is sufficient for trend identification but may not perfectly predict full-training behavior. Third, the final metric gains for paraphrase detection were small, so stronger conclusions there should be made cautiously. A hyperparameter sweep was not conducted for paraphrase detection, so we are unable to make a truly fair comparison of the two optimizers beyond the baselines.

## 7 Conclusion

Overall, SOAP shows potential as an alternative optimizer to AdamW, showing faster convergence and greater tolerance for larger learning rates. While the original SOAP paper demonstrates its potential in pretraining, this project demonstrates its value for fine-tuning as well. It is a simple change that could yield more efficient training behavior. However, its performance is highly dependent on hyperparameter selection, and the final scores are capped at a similar range to AdamW’s,

suggesting that model scaling, data scaling, and architectural changes are still required for significant improvements in final performance.

## References

- [1] Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam for language modeling. In *International Conference on Learning Representations (ICLR)*, 2025.
- [2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.
- [4] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, 2015.
- [5] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1842–1850. PMLR, 2018.
- [6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [7] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [8] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.