Black Sea Journal of Engineering and Science
doi: 10.34248/bsengineering.1739598

Open Access Journal
e-ISSN: 2619 – 8991

Research Article
Volume 9 - Issue 1: XX-XX / January 2026

THE IMPACT OF OPTIMIZER SELECTION ON TRANSFORMER PERFORMANCE: ANALYZING THE ROLE OF MODEL COMPLEXITY AND DATASET SIZE

Hilal ÇELİK[1*], Ramazan KATIRCI[1]

[1]*Sivas University of Science and Technology, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, 58000, Sivas, Türkiye*

Abstract: Model complexity, dataset size and optimizer choice critically influence machine learning model performance, especially in complex architectures like Transformers. This study aims to analyze the impact of seven optimizers —Adam, AdamW, AdaBelief, RMSprop, Nadam, Adagrad and SGD—across two Transformer configurations and three dataset sizes. Results show adaptive optimizers generally outperform non-adaptive ones like SGD, particularly as dataset size grows. For smaller datasets (20K, 50K), Adam, AdamW, Nadam and RMSprop perform best on low-complexity models, while AdaBelief, Adagrad and SGD excel with higher complexity. On the largest dataset (~140K samples), Nadam and RMSprop lead in low-complexity models, whereas Adam, AdaBelief, Adagrad, SGD and AdamW do so in high-complexity models. Notably, low-complexity models train more than twice as fast and, in some cases, achieve better accuracy and lower loss than their high-complexity counterparts. This result highlighting the importance of balancing optimizer choice, dataset size and model complexity for efficiency and accuracy. These results emphasize the trade-offs associated with optimizing model efficiency and accuracy through the interplay of optimizer selection, dataset size and model complexity.

Keywords: Transformer architecture, Optimizer comparison, Model complexity, Dataset size, Optimizers, Training efficiency

***Corresponding author:** Sivas University of Science and Technology, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, 58000, Sivas, Türkiye
E mail: hilalcelik@sivas.edu.tr (H. ÇELİK)
Hilal ÇELİK https://orcid.org/0000-0001-5428-3411
Ramazan KATIRCI https://orcid.org/0000-0003-2448-011X

Received: July 11, 2025
Accepted: November 22, 2025
Published: January 15, 2026

Cite as: Çelik, H., Katırcı, R. (2026). The impact of optimizer selection on transformer performance: Analyzing the role of model complexity and dataset size. *Black Sea Journal of Engineering and Science*, 9(1), xx–xx.

1. Introduction

Natural language processing (NLP) is a subfield of artificial intelligence that focuses on enabling machines to comprehend and process human language (Rithika, 2024). The importance of Natural Language Processing (NLP) lies in its ability to enable the implementation of applications such as machine translation (Abdulmumin et al., 2021), question-answering systems (Shoeybi et al., 2020; Çelik et al., 2024; İşlek et al., 2024), text summarization (Sun and Platoš, 2024), sentiment analysis (Mahadevaswamy and Swathi, 2022), speech recognition (Sarvepalli et al., 2015; Jurafsky and Martin, 2023), and others. Over time, however, it became clear that these techniques were inadequate. These techniques struggled to handle complex linguistic patterns, capture long-range dependencies, and process diverse language structures effectively. To overcome these challenges, Transformer-based models (Vaswani et al., 2017), particularly Generative Pre-Training (GPT) models (Radford et al., 2019; Yan and Shao, 2024), have emerged as a powerful solution. Unlike conventional models, the Transformer architecture distinguishes itself by employing attention mechanisms that allow for non-sequential, parallel processing of input data. This enables the model to efficiently capture long-range dependencies and contextual relationships across the entire sequence (Vaswani et al., 2017; Katırcı and Çelik, 2024). In addition to its ability to model long-range dependencies through parallel attention mechanisms, the Transformer architecture incorporates multiple components—such as multi-head self-attention, positional encoding and feedforward layers—that significantly enhance its representational capacity (Katırcı and Çelik, 2025a). However, the increased number of hyperparameters and architectural complexity also raise the challenges of model configuration, optimization and training time, often demanding substantial computational resources (Vaswani et al., 2017; Katırcı and Çelik, 2025b). Despite these challenges, the Transformer architecture remains a strong candidate for a wide range of NLP tasks due to its high representational capacity. Its performance, however, is highly sensitive to architectural choices and hyperparameter tuning. Although primar ily recognized for its attention mechanism, further research has enhanced other components, including positional encoding (Zheng et al., 2022; Su et al., 2024), layered structure (Razzhigaev et al., 2024) and parallel processing capabilities (Lan et al., 2020; Shoeybi et al.,

2020). Building on these developments, subsequent studies have focused on how various hyperparameters—such as model size (Zhao et al., 2024), optimizer selection (Chen et al., 2022) and dataset size variations (Zaheer and Shaziya, 2019), highlighting their impact on the efficiency and effectiveness of transformer models. For instance, Yehudai et al. (Yehudai et al., 2024) aim to calculate the frequency of each element in the input sequence by identifying its type (such as numbers or words) using the Transformer architecture. They demonstrate that while large models handle this effectively, smaller and single-layer models fail to achieve the same level of performance. Similarly, Fang et al. (2023) analyzed the impact of model depth and width on BERT and found that narrower, deeper models generally perform better on complex tasks such as question answering (QA) (Fang et al., 2023). However, performance decreases more sharply when depth is reduced than when width is decreased.

The performance of Transformer models heavily depends on the selected optimizer, as it controls learning dynamics by adjusting parameters such as weights and biases (Babaeianjelodar et al., 2020). These factors significantly impact how well a model learns and converges during training (You et al., 2020). The effect of the optimizer on Transformer performance, which is a major focus of this study, has been analyzed either individually or in limited combinations in previous research. Studies on the Transformer architecture have employed a variety of optimizers, including Adaptive Moment Estimation (Adam) (Vaswani et al., 2017), Adaptive Factorization Optimizer (Adafactor) (Shazeer and Stern, 2018), Stochastic Gradient Descent (SGD) (Jelassi and Li, 2022), Layerwise Adaptive Large Batch Optimization (LAMB) (You et al., 2020), Adaptive Moment Estimation with Weight Decay (AdamW) (Wang and Aitchison, 2024), Generalized AdaGrad (G-AdaGrad)(Chakrabarti and Chopra, 2021) and Adaptive Belief Optimization (AdaBelief) (Zhuang et al., 2020). To better understand the impact of these optimizers on transformer performance, several studies have assessed their effects in various contexts, with some specifically focusing on the performance of individual optimizers. For instance, the Adafactor optimizer was introduced by Shazeer and Stern (2018) to address memory limitations in large-scale models like Transformers. Their method significantly reduces memory usage by maintaining only per-row and per-column sums of the exponential moving averages of squared gradients. This achieves performance comparable to Adam with much less memory consumption. This approach has been widely adopted for optimizing large-scale Transformer models, particularly when working with resource-constrained environments (Shazeer and Stern, 2018). Jelassi and Li (2022) show that stochastic gradient descent (SGD) with momentum improves convergence and generalization by reducing training noise. They further demonstrate that Gradient Descent with Momentum (GD+M) outperforms Standard Gradient Descent (GD) in certain deep learning tasks. This is especially true when examples share features but show variations in classification confidence (Jelassi and Li, 2022). You et al. (2020) introduced the Layerwise Adaptive Large Batch (LAMB) optimization algorithm to enable adaptive learning rates in SGD. LAMB employs layerwise adaptive learning rates, enhancing training efficiency with large mini-batches. This method is particularly effective in reducing training time for models like BERT, without sacrificing performance (You et al., 2020). Wang and Aitchison used AdamW to relate weight decay to the exponential moving average (EMA) and enable hyperparameter tuning based on this relationship. AdamW provides critical insights on how to adjust optimal weight decay values depending on the model and dataset size (Wang and Aitchison, 2024). Chakrabarti and Chopra proposed a new fast optimizer, Generalized AdaGrad (G-AdaGrad), for accelerating the solution of potentially non-convex machine learning problems. Specifically, they adopt a state-space perspective for analyzing the convergence of gradient acceleration algorithms, namely G-AdaGrad and Adam, in machine learning. The state space models studied are governed by ordinary differential equations (Chakrabarti and Chopra, 2021). Samiksha (2025) introduced ZetA, a novel optimizer that integrates Adam's adaptive gradient updates with Riemann zeta–based dynamic scaling. Using a fully connected neural network trained on SVHN, CIFAR-10, CIFAR-100, STL-10, and noisy CIFAR-10, the study demonstrated that ZetA improves accuracy, robustness, and computational efficiency, providing a stable alternative to Adam, particularly in noisy or complex classification tasks. Vaidhyanathan et al. introduced Velocity-Regularized Adam (VRAdam), a physics-inspired optimizer that adds a velocity-based regularization term to stabilize training and prevent oscillations. Tested on CNN, Transformer, and GFlowNet architectures, VRAdam outperformed AdamW, showing improved stability and faster convergence (Vradam, 2025).

Comprehensive studies that evaluate a wide range of optimizers under diverse conditions remain scarce. Most existing research focuses on individual optimizers or a limited subset, often without considering their performance across varying scenarios. This gap highlights the need for a more extensive analysis that analyzes multiple optimizers under different conditions to gain deeper insights into their effectiveness. In this context, Zaheer et al. (2020) systematically evaluated a range of optimizers, including SGD, Nesterov Momentum, RMSProp, Adam, AdaGrad and AdaDelta, across several benchmark datasets, namely MNIST, FashionMNIST, CIFAR-10 and CIFAR-100, focusing on traditional deep learning models (Zaheer and Shaziya, 2019). In a similar study**,** Chen et al. (2024) utilized a physics-informed neural network (PINN) model and employed the optimizers PIDAO (AdSI), Adam, RMSprop**,** AdamW75, and AdaHB to solve partial differential equations, aiming

to evaluate and compare their performance in terms of convergence speed, numerical stability, and optimization accuracy in complex computational scenarios. Guan (2024) also introduces AdaPlus, an optimizer combining AdamW, Nadam, and AdaBelief with Nesterov momentum and no extra hyperparameters. Experiments using LSTM for language modeling and VGG-11, ResNet-34, and DenseNet-121 for image classification show it matches or slightly outperforms SGD with momentum and surpasses other adaptive optimizers, also demonstrating stability in GAN training (Guan, 2024). In another study, Saray and Çavdar applied various optimizers—including Nadam, Adadelta, Adamax, Adam, Adagrad, SGD, and RMSprop—to CNN models trained on the Fashion MNIST dataset. They found that Nadam and Adadelta achieved the highest accuracy, while RMSprop performed worse, showing that optimizer choice significantly affects deep learning classification performance (Saray and Çavdar, 2024). In contrast, this study uses Transformer architectures at two levels, applies the same datasets with varying sizes and evaluates total training time for each optimizer, offering a more comprehensive analysis. In addition, a study by Zhao et al. (2024) compares optimization algorithms (SGD, Adafactor, Adam, Lion and Signum) across model sizes (150M, 300M, 600M parameters) and two Transformer architectures, primarily focusing on final validation loss performance (Zhao et al., 2024). While similar in structure, this study differs by analyzing training efficiency and convergence behavior, evaluating both accuracy and loss, considering optimizer convergence and total training time for every optimizer and including a broader set of optimizers.

This study differs from previous research by systematically evaluating the impact of optimizer selection on transformer performance across two model configurations and three dataset sizes**.** Unlike most existing studies, which focus on individual optimizers or a limited subset without considering their performance across different scenarios, this study evaluates the combined effects of optimization algorithms across these model configurations and dataset scales. By incorporating training time as a key performance metric**,** alongside accuracy and efficiency measures, this study provides valuable insights into model effectiveness**,** contributing to a more comprehensive understanding of optimization strategies in Transformer architectures**.**

1.1. Impact of Optimizer Selection on Accuracy and Loss

This study demonstrates that optimizer selection significantly influences accuracy and loss, particularly as dataset size increases. For smaller datasets, complex models tend to perform worse, but this performance gap narrows with larger datasets. For some optimizers, complex models provide only marginally better results compared to simpler models.

1.2. Training Time Trends Relative to Model Complexity

This study shows that training time increases with both dataset size and model complexity across different optimizers. While training times are generally similar for smaller datasets, differences among optimizers become more apparent as dataset size and model complexity grow. Some optimizers achieve better performance with less training time when used with lower-complexity models.

1.3. Convergence Behavior Across Optimizers and Models

This study finds that in low-complexity models, all optimizers exhibit similar convergence behavior. With increasing dataset size, optimizers stabilize earlier during training. In more complex models, convergence occurs faster overall. Notably, NAdam and RMSprop show slightly more aggressive convergence, whereas SGD suffers from slower convergence and higher loss, reflecting known challenges in Transformer optimization. This study indicates that optimizer selection is the key factor influencing model performance, with dataset size and model complexity playing secondary but important roles. Increasing complexity alone does not ensure better results, especially with limited data or poor optimizer choices, whereas a well-chosen optimizer combined with a simpler model can achieve high performance more efficiently.

The structure of the paper is as follows: Section II details the methodology, including dataset selection, model architecture, hyperparameter configurations, training procedures, parallel computing techniques and model evaluation metrics. Section III presents the results and explains the reasons for comparing optimizers, model configurations, dataset sizes and training times. It discusses how these factors contribute to the performance results and how they interact with each other. Section IV provides the conclusion, summarizing the key findings. Section V presents the discussion and limitations, addressing the study's limitations and suggesting directions for future research.

2. Materials and Methods

In this study, the Transformer architecture was designed with two hyperparameter combinations, as shown in Table 1, to analyze the effects of embedding size, feed-forward layers, number of heads, number of layers and especially the optimization algorithm on learning ability and performance. The embedding size (128) and feed-forward units (256), along with the number of attention heads (4-8) and layers (3-6), were adjusted to modify the model's complexity. Other hyperparameters, including the dropout rate (0.1), ReLU activation function, batch size (32) and position coding value (10,000), were kept constant across both configurations to ensure a controlled comparison. Additionally, the study incorporated multiple optimization algorithms, including Adam, AdamW, AdaBelief, RMSprop, Nadam, Adagrad

and SGD, to systematically assess their impact on model training dynamics and performance. All optimizers employed in this study utilized the default hyperparameter configurations provided by the PyTorch implementation to ensure stability, reproducibility, and fair comparison. Specifically, parameters such as momentum, weight decay, epsilon (ε), and betas were retained at their default values for each optimizer, as these configurations have been extensively validated to yield reliable convergence and robust generalization performance across diverse neural architectures.

Table 1. Model hyperparameters and values

Hyperparameters	Low Complexity	High Complexity
Embedding Dimension	128	256
Feed Forward Units	128	256
Number of Heads	4	8
Number of Layers	3	6
Dropout Rate	0,1	0,1
Activation Function	ReLU	ReLU
Batch Size	32	32
learning_rate	0.0001	0.0001
Positional Encoding	10000	10000
Optimizer	Adam, AdamW, AdaBelief, RMSprop, Nadam, Adagrad, SGD	

Model training was performed in parallel on four GPUs using the CUDA parallel computing platform and the PyTorch library, significantly reducing training time on large datasets. CUDA enabled efficient GPU utilization for deep learning workloads and PyTorch's dynamic computation graph provided flexibility for experimenting with different hyperparameter configurations (Developer, 2025). Parallel computing with CUDA ensured scalable performance, optimizing resource utilization and improving training efficiency (Li, 2019).

The dataset used for this study comprises 140,873 Turkish-English sentence pairs. Given Turkish's morphological complexity and agglutinative structure, it represents a challenging low-resource language pair (Pan et al., 2020). To evaluate model performance across different dataset scales, the dataset was partitioned into three subsets: small (20K), medium (50K) and large (140,873). This partitioning enabled a comprehensive assessment of the effects of different optimizers and Transformer configurations on learning dynamics.

In this study, the data were randomly selected, and reproducibility was ensured by setting random_state to 42. This value is commonly adopted in the literature to maintain consistency and enable reproducibility of experimental results. The resulting dataset was divided into two subsets: 80% was used for training, and 20% was used for validation. This division facilitated model training and performance evaluation.

The model's performance was evaluated using five key metrics: training accuracy, training loss, validation accuracy, validation loss and training time. Training accuracy and loss were used to analyze the learning process, while validation accuracy and loss assessed generalization to unseen data. Additionally, training time was recorded to measure the computational efficiency of each optimizer across different dataset sizes and model configurations.

The high-performance computing (HPC) system used in this study runs on Rocky Linux 8.5 (based on Red Hat 8.5) and is powered by AMD EPYC 7543 processors. The system includes 8 compute nodes, each with 64 cores and 1024 GB of memory. For high-speed data transfer, it features HDR InfiniBand with 200 Gbps connectivity.

3. Results

This section summarizes the experimental findings, highlighting how dataset size, model complexity and optimizer choice influence Transformer performance and training efficiency.

Table 2 presents a comparative analysis of optimizer performance across different dataset sizes and model complexities. The low-complexity model configuration consists of an embedding dimension of 128, 128 feed-forward units, 4 attention heads and 3 layers, whereas the high-complexity model employs an embedding dimension of 256, 256 feed-forward units, 8 attention heads and 6 layers. Both models share the same dropout rate, activation function, batch size, positional encoding and a diverse set of optimizers. Notably, the most significant findings are highlighted in bold to emphasize key performance differences across various settings. The results demonstrate how optimization choices influence learning efficiency, with distinct patterns emerging across different complexity levels and dataset sizes. The analysis highlights how optimizer effectiveness shifts with increasing dataset size, emphasizing the trade-offs between accuracy, computational cost and generalization. These insights contribute to identifying optimal optimizer choices for different model complexities, facilitating efficient training and guiding hyperparameter tuning strategies.

For both the 20K and 50K sample datasets, the experimental results demonstrated similar performance trends across optimizers. Adam, AdamW, AdaBelief, RMSprop and Nadam achieved the best balance of accuracy, generalization and training efficiency. These optimizers consistently outperformed SGD and Adagrad, with SGD exhibiting the weakest performance, even lower than Adagrad. Notably, for both small and large datasets, low-complexity models using Adam, AdamW, RMSprop and Nadam showed higher accuracy and lower loss compared to high-complexity models. Similarly, high-complexity models using AdaBelief, Adagrad and SGD also showed higher accuracy and lower loss compared to their low-complexity counterparts. For the full dataset, unlike the 20K and 50K samples where low-complexity models with Adam, AdamW, RMSprop and

Nadam performed better, only RMSprop and Nadam maintained higher accuracy and lower loss. Similarly, high-complexity models using AdaBelief, Adagrad and SGD continued to outperform their low-complexity counterparts. Moreover, in the full dataset, all optimizers demonstrated higher accuracy and lower loss in high-complexity models compared to low-complexity models. In addition, as dataset size and model complexity increased, training time also increased. The results emphasize the importance of selecting appropriate optimizer and model configurations while considering the increasing computational costs associated with larger datasets.

For small datasets (20K samples), the experimental results in this study indicated that trend where low-complexity models tended to outperform their high-complexity counterparts in terms of both validation accuracy and loss. Among the evaluated optimizers, Nadam achieved the highest validation accuracy of 91.79% with the lowest validation loss of 0.1561, indicating its robust performance in small-data regimes. AdamW and AdaBelief also yielded strong results, with validation accuracies of 91.47% and 91.62% and validation losses of 0.1643 and 0.1592, respectively. These optimizers appear to balance model training and generalization effectively in environments where data is limited. On the other hand, SGD performed poorly, achieving only 83.52% validation accuracy and exhibited the highest loss, underscoring its ineffectiveness in small-data contexts.

Table 2. Performance comparison of seven optimization algorithms (Adam, AdamW, AdaBelief, RMSprop, Nadam, Adagrad and SGD) across different dataset sizes (20K, 50K, and ~140K samples) and model complexities (low vs. high)

		Low complexity					High complexity				
Data Samples	Optimizer	Train Accuracy	Train Loss	Valid Loss	Valid Accuracy	Time	Train Accuracy	Train Loss	Valid Loss	Valid Accuracy	Time
20K data samples	Adam	91.81	0.1581	0.1634	91.52	0.06.16	89.40	0.1894	0.1882	89.49	0.15.16
	AdamW	91.87	0.1578	0.1643	91.47	0.06.11	89.42	0.1871	0.1850	89.34	0.15.12
	AdaBelief	91.53	0.1589	0.1592	91.62	0.06.40	93.38	0.1170	0.1173	93.44	0.16.41
	RMSprop	91.69	0.1586	0.1562	91.75	0.06.10	89.99	0.1791	0.1805	89.95	0.15.45
	Nadam	91.87	0.1555	0.1561	91.79	0.06.03	83.37	0.3796	0.3837	83.27	0.15.42
	Adagrad	87.61	0.5564	0.5479	87.87	0.05.27	88.21	0.2829	0.2840	88.22	0.15.17
	SGD	83.30	0.4791	0.4735	83.52	0.06.10	83.90	0.3853	0.3914	83.59	0.14.27
50K data samples	Adam	93.73	0.1115	0.1138	93.63	0.14.38	92.79	0.1108	0.1103	92.79	0.37.40
	AdamW	93.88	0.1090	0.1085	93.78	0.15.05	92.14	0.1210	0.1238	91.96	0.38.02
	AdaBelief	94.05	0.0978	0.1005	93.87	0.15.33	96.37	0.0513	0.0513	96.40	0.40.13
	RMSprop	93.56	0.1206	0.1211	93.43	0.14.28	92.95	0.1357	0.1328	93.07	0.39.40
	Nadam	93.79	0.1088	0.1074	93.94	0.13.52	86.97	0.2657	0.2625	87.10	0.37.38
	Adagrad	87.66	0.4382	0.4362	87.67	0.14.06	88.79	0.2594	0.2605	88.72	0.36.22
	SGD	83.35	0.4449	0.4492	83.27	0.14.18	86.94	0.3669	0.3729	86.77	0.35.42
~140Kdata samples	Adam	98.71	0.0188	0.0186	98.74	0.51.51	98.91	0.0133	0.0133	98.90	2.58.48
	AdamW	98.70	0.0189	0.0187	98.73	0.50.26	98.98	0.0126	0.0126	98.96	2.59.53
	AdaBelief	98.78	0.0165	0.0159	98.81	0.54.01	99.47	0.0061	0.0061	99.46	3.17.49
	RMSprop	98.56	0.0267	0.0261	98.59	0.51.07	98.41	0.0361	0.0360	98.41	3.00.53
	Nadam	98.73	0.0179	0.0181	98.72	0.50.54	98.37	0.0212	0.0208	98.39	3.21.32
	Adagrad	96.28	0.1318	0.1315	96.28	0.49.59	96.75	0.0772	0.0776	96.72	2.52.26
	SGD	95.00	0.1415	0.1419	94.99	0.51.39	96.02	0.1182	0.1182	96.05	2.51.18

Adam = adaptive moment estimation, AdamW = adam with weight decay, AdaBelief = adaptive belief optimizer, RMSprop = root mean square propagation, Nadam = nesterov-accelerated, Adagrad = adaptive gradient algorithm, SGD = stochastic gradient descent.

For medium-sized datasets (50K samples), the results show that low-complexity models tend to perform better with optimizers like Adam, AdamW, RMSprop and Nadam, while higher complexity models benefit from optimizers like AdaBelief, Adagrad and SGD. Adam, AdamW, RMSprop and Nadam demonstrate better performance in simpler models, achieving relatively good validation accuracy with moderate loss values. However, in high-complexity settings, models trained with AdaBelief, Adagrad and SGD outperform the others, as

they can handle the increased model complexity more effectively. Specifically, AdaBelief achieves a high train accuracy of 94.05%, outperforming Adam and AdamW, which have train accuracies of 93.74% and 93.88%, respectively, demonstrating the effectiveness of the optimizer even with a low-complexity model. Meanwhile, Nadam performs well in low-complexity scenarios, but it faces challenges with larger models, showing a drop in validation accuracy and an increase in validation loss. Adagrad and SGD exhibit lower performance on medium-sized datasets, achieving poorer accuracy and higher loss when compared to other optimizers, which perform significantly better across both low and high-complexity models. These results emphasize that optimizers like AdaBelief, Adagrad and SGD are more effective in high-complexity models, while Adam, AdamW RMSprop and Nadam perform better with simpler models on smaller datasets. These findings highlight the importance of aligning optimizer selection with both dataset size and model complexity, as adaptive optimizers like Adam and AdamW excel in less complex models, whereas AdaBelief and SGD manage larger parameter spaces effectively in more complex models.

For large datasets (140,873 samples), high-complexity models consistently outperform low-complexity models, achieving the highest validation accuracy, which highlights the benefits of increased model complexity. Among the optimizers, AdaBelief achieved the best performance, with a validation accuracy of 99.46% and the lowest validation loss of 0.0061, closely followed by AdamW with 98.96% accuracy and a loss of 0.0126. In contrast, SGD performed poorly, reaching only 96.05% accuracy and exhibiting the highest loss, making it less suitable for large datasets. These results demonstrate that pairing high-complexity models with advanced optimizers such as AdaBelief, Adam, or AdamW is most effective for achieving superior accuracy and generalization. The advantage of these optimizers can be attributed to their adaptive learning mechanisms, which enable efficient gradient updates and better convergence. Conversely, the weaker performance of SGD, likely due to its inability to dynamically adapt learning rates, underscores the importance of selecting optimizers that fully leverage model capacity for optimal results.

The relationship between model complexity and dataset size significantly influences optimizer performance in this study. When trained on smaller datasets (20K and 50K samples), models utilizing Adam, AdamW, RMSprop and Nadam show a decline in both training and validation accuracy as model complexity increases. This pattern is consistent with the general understanding that higher model complexity worsens performance problems in data-limited scenarios, making these optimizers less effective. However, AdaBelief, Adagrad and SGD demonstrate robustness with smaller datasets, achieving higher accuracy and lower loss as model complexity increases. In contrast, when the dataset size increases to 140,000 samples, Adam, AdamW, AdaBelief, Adagrad and SGD benefit from larger model architectures, achieving higher performance, while RMSprop and Nadam experience a decline in accuracy. These results can be explained by the optimizers' inherent characteristics. Adaptive optimizers like Adam and AdamW tend to overfit on smaller datasets due to aggressive learning rate adjustments, while SGD and Adagrad generalize better by avoiding excessive updates. AdaBelief's stability makes it effective across all dataset sizes. RMSprop and Nadam's decline in performance, even with larger datasets, may stem from their sensitivity to hyperparameters and difficulty in scaling with model complexity. This highlights the need to match optimizer choice with dataset size and model architecture.

AdaBelief consistently excelled with high-complexity models and large datasets because its belief-based adaptive mechanism dynamically adjusts learning rates based on gradient reliability rather than magnitude. This enables more stable and efficient convergence in deep architectures with large parameter spaces, reducing gradient noise and improving generalization (Zhuang et al., 2020; Zhang et al., 2022).

RMSprop and Nadam perform better as model complexity increases. In large deep learning models, the accumulation of gradient variance can destabilize exponential moving averages, leading to overcorrection in adaptive updates and reduced training stability.(Tomihari and Sato, 2025).

SGD demonstrated consistent weakness across all dataset scales, as its fixed learning rate and non-adaptive nature make it prone to slow convergence and difficulty escaping local minima. This limitation becomes more pronounced in high-dimensional Transformer architectures, where adaptive optimizers can better adjust to complex gradient dynamics (Choi et al., 2019; Pan and Li, 2023; Zhang et al., 2024; Tomihari and Sato, 2025).

This study primarily focuses on the impact of dataset size and model complexity on accuracy and loss across various optimization algorithms, while also providing a comprehensive assessment of their influence on training time. Consequently, as dataset size and model complexity increase, training times rise significantly, further emphasizing the need to balance performance and computational efficiency. For instance, with 20K data points, Adam achieves 91.81% training accuracy in 6.16 minutes for a low-complexity model, whereas the same dataset requires 15.16 minutes for a high-complexity model, with accuracy dropping to 89.40%. Similarly, for 50K data points, Adam attains 93.73% accuracy in 14.38 minutes for a low-complexity model, whereas it takes 37.40 minutes to reach 92.79% accuracy in a high-complexity model. This trend becomes even more pronounced with the full dataset (140K), where Adam achieves 98.71% accuracy in 51.51 minutes for a low-complexity model, while the high-complexity model requires 2 hours, 58 minutes and 48 seconds to reach 98.91% accuracy. Among the optimization algorithms,

AdaBelief achieves the highest accuracy (99.47%) in high-complexity models with the full dataset but requires an extended training time of 3.17 hours. In contrast, optimizers such as Nadam and Adagrad yield faster results in low-complexity models (e.g., ~6 minutes for 20K data). High-complexity models generally yield better performance but require significantly longer training times, especially for larger datasets. For example, training a high-complexity model on the full dataset takes over three hours, compared to just six minutes for a low-complexity model on a smaller dataset. This highlights the need for efficient optimization strategies that can balance performance and computational cost. While time-intensive algorithms are preferable for achieving higher accuracy, faster optimizers may be more suitable for time-constrained applications. Additionally, the findings underscore the direct impact of dataset size and model complexity on computational cost. Furthermore, specific optimization techniques or hyperparameter adjustments might enable higher accuracy within shorter training durations. This study also emphasizes the trade-off between model complexity and training efficiency.

3.1. Evaluation of Loss Across Models, Optimizers and Data Size

The effectiveness of an optimization algorithm is closely linked to how efficiently it minimizes both training and validation loss during training. Understanding the loss dynamics across different model complexities and dataset sizes provides valuable insights into optimizer performance. For this reason, this section assesses how various optimizers influence the reduction of training and validation loss, shedding light on their convergence behavior and effectiveness in different scenarios. Figure 1 display the training and validation loss for different optimizers across epoch values, using Transformer architecture with hyperparameters tailored for both low and high complexity models. The graphs on the left (Figures a, c and e) show the low-complexity model with 128 embedding dimensions, 4 attention heads and 3 layers, while the graphs on the right (Figures b, c and d) illustrate the high-complexity model with 256 embedding dimensions, 8 attention heads and 6 layers. A key factor in model training is how optimization algorithms impact the reduction of both training and validation loss over time. In both sets of graphs, the training and validation losses generally decrease as the number of epochs increases, suggesting that the model continues to learn from the data. While the loss decreases for all cases, the rate of reduction varies across optimizers, with some showing faster decreases, indicating distinct convergence behaviors. The varying rates of loss reduction across optimizers suggest differences in their efficiency and adaptability, with some achieving faster convergence, which could imply better generalization.

The results presented in Figure 1a and Figure 1b illustrate the training and validation losses across different optimization algorithms for a dataset comprising 20K samples, evaluated under two distinct model architectures. Among the evaluated optimizers, Adam, AdamW, Nadam and RMSprop exhibit rapid convergence and maintain low loss values, particularly in high-complexity models. While AdaBelief initially shows higher loss, it quickly adapts and achieves comparable or superior final performance within the same number of epochs. This effect is especially pronounced in more complex architectures, where AdaBelief's adaptive mechanism enhances stability and generalization. By dynamically adjusting learning rates based on gradient predictability, AdaBelief effectively mitigates sharp loss fluctuations, leading to improved optimization efficiency in deeper models with larger parameter spaces (Chakrabarti and Chopra, 2021). Additionally, SGD and Adagrad exhibit distinctive performance characteristics compared to the other optimizers considered in this study. SGD demonstrates slower convergence, resulting in higher final loss values across nearly all experimental conditions. In contrast, Adagrad starts with loss values similar to those of Adam, AdamW, Nadam and RMSprop but gradually aligns more with SGD's performance, ultimately resulting in higher loss values. The performance disparity between SGD and Adagrad is influenced by factors such as model complexity and dataset size, suggesting that their effectiveness depends on these variables. The slower convergence and higher final loss values of SGD, as observed in this study, can be attributed to its sensitivity to the learning rate and the absence of adaptive adjustments. SGD typically requires careful tuning of the learning rate and hyperparameters like momentum to prevent slow convergence and local minima issues. When these factors are not optimally set, SGD struggles to converge efficiently, particularly with complex models or large datasets (Fehrman and Gess, 2020). On the other hand, Adagrad starts with comparable loss values to Adam and other optimizers but eventually suffers from diminishing updates. While its adaptive learning rate initially provides an advantage, over time, the accumulated sum of squared gradients leads to excessively small updates, resulting in slower convergence and higher final loss values (Luo, 2019).

As illustrated in Figure 1 (subplots a–f), this analysis focuses on the training and validation losses across different dataset sizes and model complexities. The results presented in Figure 1c and 1d illustrate the training and validation losses for a dataset size of 50K samples across two model architectures. Similar to the 20K dataset (Figure 1a and 1b), Adam, Adam-W and Nadam demonstrate rapid convergence and maintain low loss values, with Adam-W showing the most stable performance. RMSprop also converges quickly but exhibits minor fluctuations in validation loss, indicating slight instability. Adagrad and SGD show slower convergence and higher loss values, with SGD performing the worst. As model complexity increases, Adam-based optimizers remain the most effective, while RMSprop, despite starting with a very low loss, may require careful

tuning due to the possibility of progressing too quickly without sufficient learning. These observations emphasize the critical role of optimizer selection in balancing convergence speed, stability and performance, particularly as model architectures grow in complexity. Compared to the 20K-sample case, the simpler model (low complexity) trained with 50K samples exhibits higher initial loss but achieves lower final loss, suggesting that larger datasets introduce variability in early optimization but improve generalization. For the more complex model, increasing dataset size leads to lower initial and final loss, indicating that larger datasets allow complex models to utilize their capacity more effectively. This effect is particularly pronounced in complex models, where increased data availability helps maximize representational capacity. In contrast to 20K and 50K samples, Figure 1e and Figure 1f (140,873 samples) show that models trained on larger datasets start with lower initial loss and achieve even lower final loss, highlighting the benefits of larger datasets for providing a richer learning signal, improving generalization and enhancing stability across all optimizers. These results underscore the importance of selecting the right optimizer, particularly as architectural complexity increases.

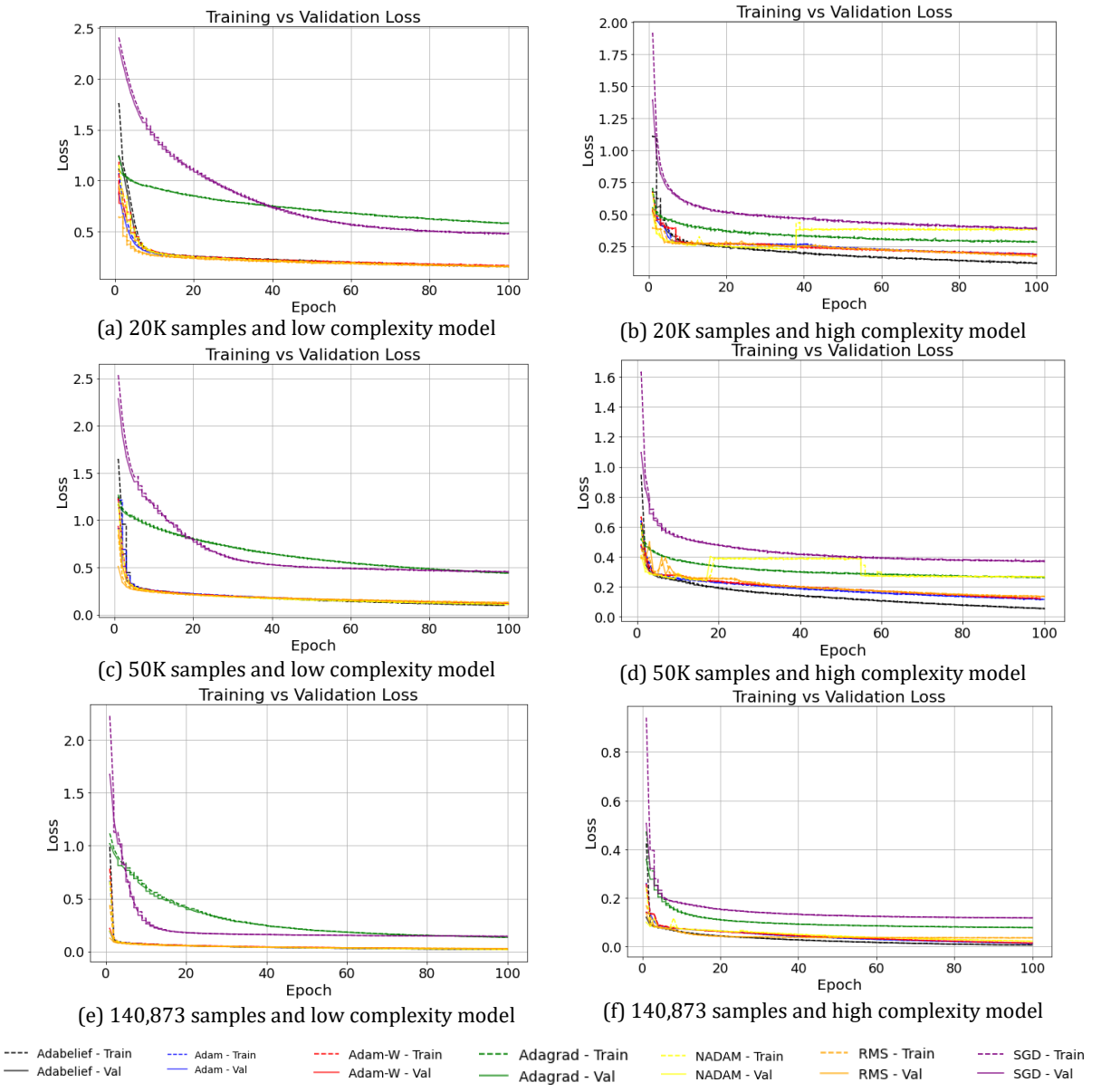


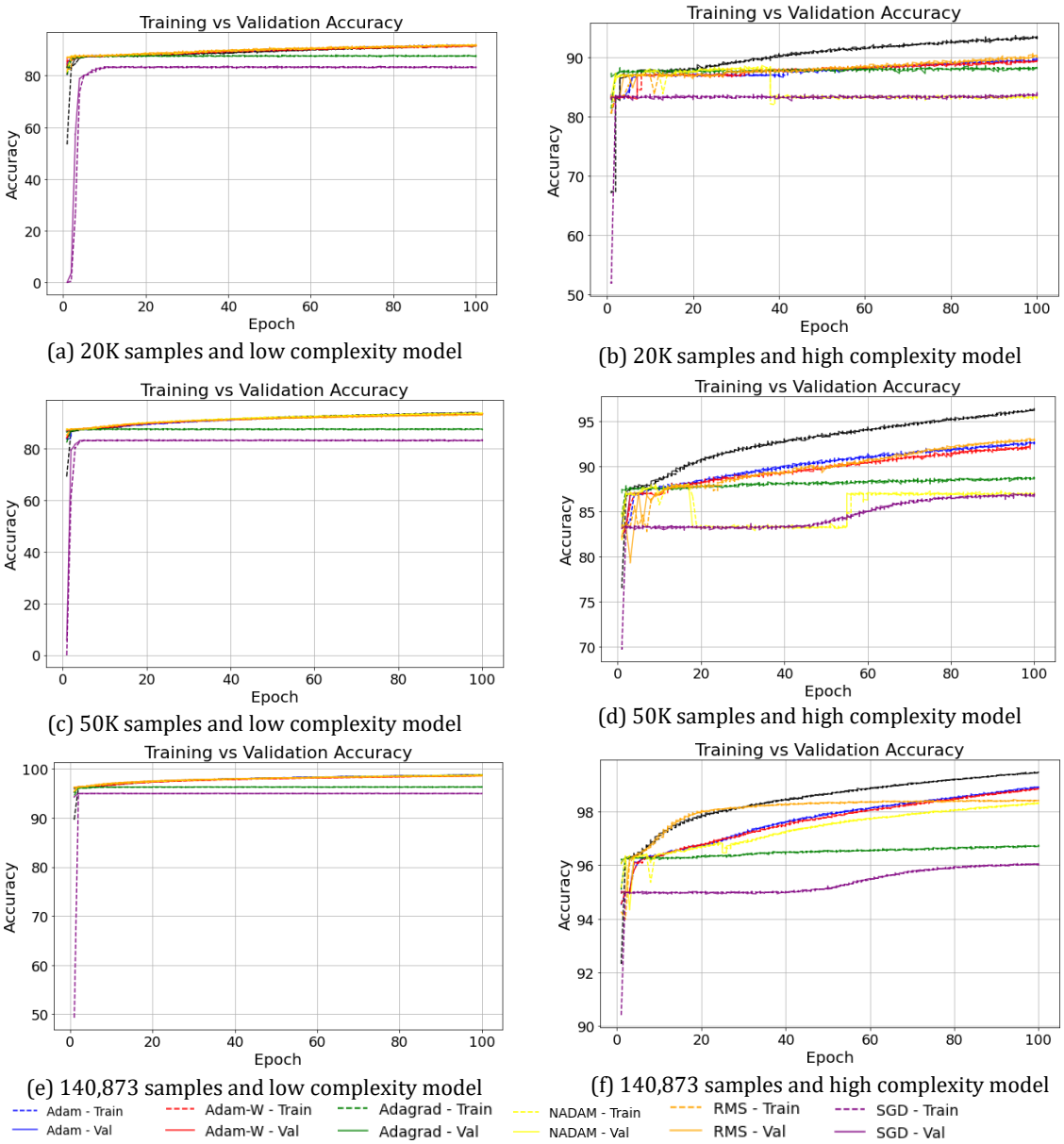
Figure 1. Training and validation loss per epoch for different optimizers across dataset sizes (20K, 50K, 140,873) and model complexities (low and high).

The superior performance of Adam-based optimizers can be attributed to their adaptive learning rate mechanisms, which efficiently navigate complex loss landscapes, especially with smaller datasets. In contrast, SGD and Adagrad face difficulties with sparse or noisy gradients, leading to slower convergence and higher final loss values, particularly with larger or more complex models. This suggests that for smaller datasets, adaptive

optimizers are not only preferable but essential for stable and efficient training. Additionally, combining adaptive optimizers with regularization techniques, such as dropout or weight decay, could further improve performance by preventing overfitting and enhancing generalization.

3.2. Evaluation of Accuracy Across Models, Optimizers and Data Size

This subsection assesses the impact of optimization algorithms on accuracy across different model complexities and dataset sizes, focusing on learning stability, convergence speed and generalization performance. Figure 2 illustrates the accuracy of various optimizers during training for both low-complexity (smaller) and high-complexity (larger) models. The left-side graphs (Figures 2a, 2c and 2e) correspond to the smaller model, while the right-side graphs (Figures 2b, 2d and 2f) represent the larger model. Figures 2a–2b, 2c–2d and 2e–2f correspond to datasets containing 20K, 50K and 140,873 samples, respectively. These plots show accuracy trends, emphasizing the effects of model complexity and dataset size on optimization efficiency and generalization.



(a) 20K samples and low complexity model

(b) 20K samples and high complexity model

(c) 50K samples and low complexity model

(d) 50K samples and high complexity model

(e) 140,873 samples and low complexity model

(f) 140,873 samples and high complexity model

Figure 2. Training and validation accuracy per epoch for different optimizers across dataset sizes (20K, 50K, 140,873) and model complexities (low and high).

As model complexity and dataset size increase, the accuracy values of the optimization algorithms show higher initial and final performance levels, with the differences in accuracy across optimizers becoming more pronounced. These disparities are further accentuated with larger data size, highlighting the greater impact of dataset size on optimization performance. This could be due to the increased availability of data allowing the

model to better generalize, which in turn enables the optimization algorithms to perform more effectively. Additionally, larger models with more parameters tend to benefit more from larger datasets, as they require more data to fully capture the underlying patterns and avoid overfitting.

In this study, among the optimization algorithms, Adam, AdamW, AdaBelief, Nadam and RMSprop achieve high accuracy with minimal loss between training and validation values, demonstrating strong generalization and robust performance across different dataset sizes and model complexities. In contrast, SGD performs the worst, with the lowest accuracy and highest loss between training and validation results, while Adagrad follows, showing slightly better performance than SGD. This may suggest that hyperparameter optimization, particularly adjusting learning rate, momentum and weight decay, could help improve the performance of both SGD and Adagrad, as these algorithms may struggle with convergence and generalization compared to more adaptive optimizers. Nadam, on the other hand, may struggle with stabilizing more complex models due to the increased number of parameters and layers, which make it harder to adapt learning rates effectively. The presence of gradient noise in these models may further complicate training, potentially leading to slower convergence and poorer generalization.

When the model complexity was low, the optimization algorithms exhibited similar convergence behavior, suggesting that simpler models allowed all optimizers to perform effectively. However, as model complexity increased, particularly with larger dataset sizes, notable differences in convergence patterns emerged. This observation aligns with prior research, which analyzed the computational cost of deep learning models and demonstrated that as model complexity grows, convergence rates among different optimizers begin to diverge, highlighting the necessity of adaptive optimization methods for training deeper networks efficiently (Ahmad and Al-ramahi, 2023; Baskakov, 2023).

Adam, Adagrad, AdamW, RMSprop and Nadam, exhibited similar convergence rates, indicating their efficiency in adapting to increased model complexity, while Adagrad remained slightly behind toward the final stages. On the other hand, Nadam, Adam, AdamW and RMSprop, despite similar convergence trends, initially struggled to stabilize, suggesting that complex models challenge their learning rate adaptation in early training stages. Meanwhile, SGD and AdaBelief started with lower accuracy values and converged more quickly, with SGD starting from an even lower value but achieving a faster convergence. SGD's poor performance is likely due to its fixed learning rate, making it highly sensitive to hyperparameter tuning and potentially prone to slow convergence in complex models. This behavior could be attributed to the differences in how these optimizers handle gradient updates and learning rate adjustments. Adam, AdaBelief and Adagrad use adaptive learning rates, allowing them to adjust more effectively to varying gradient magnitudes, leading to more stable convergence. Nadam, AdamW and RMSprop, while also adaptive, may experience instability due to aggressive updates early in training (Zhou et al., 2021; Guan, 2024).

4. Discussion

The results demonstrate that optimizer selection plays a decisive role in performance-efficiency trade-offs across varying model complexities and dataset sizes. AdaBelief, Adagrad, and SGD performed notably better with high-complexity models, particularly as the dataset size increased. In our experiments, AdaBelief stood out by achieving promising results, including a validation accuracy of 99.46% on the full dataset with the high-complexity model. However, this performance gain came with a significant increase in training time, especially for more complex models and larger datasets. These findings highlight that while certain optimizers can better leverage increased capacity and data volume, they also introduce higher computational costs.

Interestingly, low-complexity models trained with effective optimizers such as AdaBelief or RMSprop on large datasets achieved accuracy levels comparable to those of high-complexity models. This indicates that increasing model depth does not always yield better outcomes, especially when optimizer selection or data volume is suboptimal. On smaller datasets (20K and 50K), Adam, AdamW, RMSprop, and Nadam provided better performance with low-complexity models while requiring significantly less training time. On the full dataset, Adam and AdamW showed slightly better performance on high-complexity models, whereas RMSprop and Nadam achieved marginally better results on low-complexity models. Considering that training time nearly doubles when model complexity increases within the same dataset size, these results emphasize that the trade-off between model depth and efficiency becomes especially relevant in large-scale applications. Therefore, under resource constraints, pairing effective optimizers with less complex models can offer a more practical and time-efficient solution without substantial performance loss.

5. Conclusion

In general, the findings of this study indicate that the choice of optimizer should be aligned with both dataset size and model complexity. For smaller datasets or low-complexity models, optimizers such as SGD, Nadam, RMSprop, Adam, and AdamW are more effective due to their fast convergence and lower computational cost. In contrast, Adagrad and AdaBelief are better suited for high-complexity models trained on limited data, as their adaptive mechanisms help stabilize learning. When large datasets are available, all optimizers yield higher accuracy with complex models, demonstrating that

sufficient data enables these architectures to fully exploit their representational capacity.

The findings from this study suggest that optimizer selection is a key factor influencing model performance, with dataset size and model complexity playing secondary but important roles. Increasing complexity alone does not ensure better results, especially with limited data or poor optimizer choices, whereas a well-chosen optimizer combined with a simpler model can achieve high performance more efficiently. Moreover, shorter training times can be achieved with certain optimizers on low-complexity models when dataset sizes are large, highlighting that the right combination of optimizer, dataset size and model hyperparameters enables both time efficiency and high accuracy. This emphasizes that the optimal combination of optimizer, dataset size and model hyperparameters is crucial for balancing training time and performance.

Future research could examine more optimizers and a wider range of dataset sizes to better understand their effects on model performance. Analyzing various Transformer complexity levels may offer deeper insights into efficiency-accuracy trade-offs. Additionally, exploring techniques such as model pruning or knowledge distillation could help reduce computational costs without sacrificing accuracy. Extending the study to larger datasets and real-world scenarios would further improve the generalizability of the results. Furthermore, a key limitation of this study is that each configuration was executed only once, preventing statistical validation of performance differences. Future studies should conduct multiple runs with different random seeds and apply statistical tests to ensure robustness.

Author Contributions

The percentages of the authors' contributions are presented below. All authors reviewed and approved the final version of the manuscript.

	H.C.A.	B.C.
C	50	50
D	50	50
S		100
DCP	80	20
DAI	50	50
L	50	50
W	60	40
CR	10	90
SR	50	50
PM	20	80

C=Concept, D= design, S= supervision, DCP= data collection and/or processing, DAI= data analysis and/or interpretation, L= literature search, W= writing, CR= critical review, SR= submission and revision, PM= project management.

Conflict of Interest

The authors declared that there is no conflict of interest.

Ethical Consideration

Ethics committee approval was not required for this study because of there was no study on animals or humans.

Acknowledgements

This work has been supported by the Scientific Research ProjectsCoordination Unit of the Sivas University of Science and Technology. ProjectNumber: 2024-DTP-Müh-0004. Computing resources used in this work were providedby the National Center for High Performance Computing of Türkiye (UHeM) undergrant number 5020092024. The research utilized computational resources providedby the TUBITAK ULAKBIM High Performance and Grid Computing Center (TRUBA) andthe Lütfi Albay Artificial Intelligence and Robotics Laboratory at Sivas Universityof Science and Technology."

References

Abdulmumin, I., Galadanci, B. S., & Isa, A. (2021). Enhanced back-translation for low resource neural machine translation using self-training. *Communications in Computer and Information Science, 1350*, 355–371. https://doi.org/10.1007/978-3-030-69143-1_28

Ahmad, R., & Al-Ramahi, I. A. M. (2023). Optimization of deep learning models: Benchmark and analysis. *Advances in Computational Intelligence, 3*(2), 1–15. https://doi.org/10.1007/s43674-023-00055-1

Babaeianjelodar, M., Lorenz, S., Gordon, J., Matthews, J., & Freitag, E. (2020). Quantifying gender bias in different corpora. *Companion Proceedings of the Web Conference 2020*, 752–759. https://doi.org/10.1145/3366424.3383559

Baskakov, D. (2023). *The computational complexity of machine learning* (Issue January). Springer. https://doi.org/10.1007/978-981-33-6632-9

Çelik, H., Katırcı, R., & İşlek, B. (2024). Effect of parameters on performance in question-answer model with simple RNN deep learning method. *International Conference on Scientific and Innovation Research*, 161–169.

Chakrabarti, K., & Chopra, N. (2021). Generalized AdaGrad (G-AdaGrad) and Adam: A state-space perspective. *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 1496–1501. https://doi.org/10.1109/CDC45484.2021.9682994

Chen, Y., Song, X., Lee, C., Wang, Z., Zhang, Q., Dohan, D., Kawakami, K., Kochanski, G., Doucet, A., Ranzato, M., Perel, S., & de Freitas, N. (2022). Towards learning universal hyperparameter optimizers with transformers. *Advances in Neural Information Processing Systems, 35*, 1–16.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. *arXiv*. https://arxiv.org/abs/1910.05446

Developer, N. (2025). CUDA zone. https://developer.nvidia.com/cuda-zone

Fang, H., Lee, J. U., Moosavi, N. S., & Gurevych, I. (2023). Transformers with learnable activation functions. *Findings of the EACL 2023*, 2337–2353. https://doi.org/10.18653/v1/2023.findings-eacl.181

Fehrman, B., & Gess, B. (2020). Convergence rates for the

stochastic gradient descent method for non-convex objective functions. *arXiv:1904.01517*.

Guan, L. (2024). Adaplus: Integrating Nesterov momentum and precise stepsize adjustment on AdamW basis. *ICASSP 2024*, 5210–5214. https://doi.org/10.1109/ICASSP48485.2024.10447337

İşlek, B., Katırcı, R., & Çelik, H. (2024). Enhancing question answering systems through optimal hyperparameter tuning in GRU. *8th International Artificial Intelligence and Data Processing Symposium (IDAP 2024).* https://doi.org/10.1109/IDAP64064.2024.10710732

Jelassi, S., & Li, Y. (2022). Towards understanding how momentum improves generalization in deep learning. *Proceedings of Machine Learning Research, 162*, 9965–10040.

Jurafsky, D., & Martin, J. H. (2008). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (2nd ed.). Prentice Hall.

Katırcı, R., & Çelik, H. (2024). Transformer mimarisi. https://doi.org/10.5281/zenodo.13971609

Katırcı, R., & Çelik, H. (2025a). Evaluating the impact of activation functions on transformer architecture performance. *International Science and Art Research Center*, 626–639.

Katırcı, R., & Çelik, H. (2025b). Learning rate sensitivity in transformer models: A case study in neural machine translation. https://doi.org/10.5281/zenodo.15769066

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2020). ALBERT: A lite BERT for self-supervised learning of language representations. *8th International Conference on Learning Representations (ICLR 2020).*

Li, S. (2019). Getting started with distributed data parallel. https://pytorch.org/tutorials/intermediate/ddp_tutorial.html

Luo, L. (2019). Adaptive gradient methods with dynamic bound of learning rate. *arXiv*. https://arxiv.org/abs/1902.09843

Mahadevaswamy, U. B., & Swathi, P. (2022). Sentiment analysis using bidirectional LSTM network. *Procedia Computer Science, 218*, 45–56. https://doi.org/10.1016/j.procs.2022.12.400

Pan, Y., Li, X., Yang, Y., & Dong, R. (2020). Morphological word segmentation on agglutinative languages for neural machine translation. *arXiv:2001.01589.*

Pan, Y., & Li, Y. (2023). Toward understanding why Adam converges faster than SGD for transformers. *arXiv:2306.00204*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Technical Report*.

Razzhigaev, A., Mikhalchuk, M., Goncharova, E., Oseledets, I., Dimitrov, D., & Kuznetsov, A. (2024). The shape of learning: Anisotropy and intrinsic dimensions. *arXiv.*

Rithika. (2024). Recent advances in large language models: An upshot. *Journal*, *5*(6), 137–143. https://doi.org/10.55248/gengpi.5.0624.1403

Saray, U., & Çavdar, U. (2024). Comparison of different optimization algorithms in the Fashion MNIST dataset. *International Journal of Management Science and Information Technology,* *8*(2), 52–58. https://doi.org/10.36287/ijmsit.8.2.1

Sarvepalli, S. K., Sarat, S., & Sarvepalli, K. (2015). Deep learning in neural networks: The science behind an artificial brain. https://doi.org/10.13140/RG.2.2.22512.71682

Shazeer, N., & Stern, M. (2018). Adafactor: Adaptive learning rates with sublinear memory cost. *ICML 2018*, 7322–7330.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2020). Megatron-LM: Training multi-billion parameter language models using model parallelism. https://arxiv.org/abs/1909.08053

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., & Liu, Y. (2024). RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing, 568*, Article 127063. https://doi.org/10.1016/j.neucom.2023.127063

Sun, Y., & Platoš, J. (2024). Abstractive text summarization model combining a hierarchical attention mechanism and multiobjective reinforcement learning. *Expert Systems with Applications, 248*. https://doi.org/10.1016/j.eswa.2024.123356

Tomihari, A., & Sato, I. (2025). Understanding why Adam outperforms SGD: Gradient heterogeneity in transformers. *arXiv:2502.00213.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5999–6009.

Vradam, V. A. (2025). A physics-inspired optimizer: Velocity regularized Adam. (Eksik: Yayın türü)

Wang, X., & Aitchison, L. (2024). How to set AdamW's weight decay as you scale model and dataset size. *arXiv:2405.13698.*

Yan, H., & Shao, D. (2024). Enhancing transformer training efficiency with dynamic dropout. *arXiv:2411.03236.*

Yehudai, G., Kaplan, H., Ghandeharioun, A., Geva, M., & Globerson, A. (2024). When can transformers count to n? *arXiv:2407.15160.*

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., & Hsieh, C. J. (2020). Large batch optimization for deep learning: Training BERT in 76 minutes. *ICLR 2020, 1-15.*

Zaheer, R., & Shaziya, H. (2019). A study of the optimization algorithms in deep learning. *Proceedings of the ICISC 2019*, 536–539. https://doi.org/10.1109/ICISC44355.2019.9036442

Zhang, G., Niwa, K., & Kleijn, W. B. (2022). A DNN optimizer that improves over AdaBelief by suppression of the adaptive stepsize range. *arXiv:2203.13273.*

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., & Luo, Z. (2024). Why transformers need Adam: A Hessian perspective. *NeurIPS.*

Zhao, R., Morwani, D., Brandfonbrener, D., Vyas, N., & Kakade, S. (2024). Deconstructing what makes a good optimizer for language models. *arXiv:2407.07972.*

Zheng, J., Rezagholizadeh, M., & Passban, P. (2022). Dynamic position encoding for transformers. *Proceedings of COLING 2022*, 5076–5084.

Zhou, Y., Huang, K., Cheng, C., Wang, X., Hussain, A., & Liu, X. (2021). FastAdaBelief: Improving convergence rate for belief-based adaptive optimizers by exploiting strong convexity. *arXiv:2104.13790.*

Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., & Dvornek, N. (2020). AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients. *arXiv:2010.07468.*