

Review Article

Advancements in AI-Based Compiler Optimization Techniques for Machine Learning Workloads

Vasuki Shankar¹ 

¹Nvidia Corporation, Bengaluru, Karnataka, India

Corresponding Author: 

Received: 22/Jan/2025; Accepted: 24/Feb/2025; Published: 31/Mar/2025. DOI: <https://doi.org/10.26438/ijcse/v13i3.7077>



Copyright © 2025 by author(s). This is an Open Access article distributed under the terms of the [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited & its authors credited.

Abstract: This paper primarily explores the application of AI-driven compiler optimization techniques for machine learning (ML) workloads, with a focus on reinforcement learning and neural architecture search. It examines the performance of traditional compilers compared to AI-optimized compilers leveraging various ML models, including CNNs, RNNs, FNNs, and transformers. The results indicate that AI-driven compilers — particularly those using a hybrid RL + NAS approach—outperforms traditional compilers in energy consumption, memory usage, execution time and hardware utilization. Additionally, the findings suggest that AI-based optimization techniques can streamline ML pipeline development, enhancing efficiency and performance for both resource-constrained environments and large-scale applications.

Keywords: AI-based compilers, reinforcement learning, neural architecture search, machine learning, compiler optimization.

1. Introduction

In recent decades, ML applications have experienced rapid growth, placing increasing demands on computing systems in terms of performance and efficiency. Both training and inference in machine learning require vast computational resources, from classical algorithms to deep neural networks. As model complexity increases, so does the need for optimization techniques that can efficiently utilize the underlying hardware. Compiler optimization has long been a crucial aspect of translating high-level code into machine instructions to enhance software performance. However, traditional compiler optimization techniques often fall short in addressing the unique computational patterns and memory access behaviors of ML workloads. To overcome these challenges, recent advancements have incorporated artificial intelligence (AI) into compiler optimization.

Traditionally, compilers apply a range of optimization techniques to reduce execution time, minimize memory consumption, and improve parallelism. Common approaches include loop unrolling, instruction scheduling, inlining, and vectorization. While effective for general-purpose applications, these techniques struggle to optimize ML workloads, which often involve large-scale data processing, irregular computation patterns, and complex tensor operations. Efficient optimization is especially critical for maximizing the performance of modern accelerators such as

Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and specialized AI chips. Additionally, ML workloads are dynamic in nature, with model structures and behaviors varying significantly based on the task, further complicating optimization efforts.

To address these challenges, AI-based compiler optimization techniques have emerged as a promising solution. By leveraging AI and ML algorithms, compilers can adaptively learn and apply optimization strategies tailored to specific workloads. AI-driven approaches, such as Reinforcement Learning (RL) and Neural Architecture Search (NAS), enable compilers to explore a wide range of optimization possibilities beyond rule-based heuristics. These techniques allow for more intelligent and context-aware optimizations, leading to significant improvements in execution efficiency, resource utilization, and overall system performance.

1.1 Aim and Objectives

This paper aims to identify and analyze advancements in AI-based compiler optimization techniques aimed at enhancing the performance and efficiency of artificial intelligence and machine learning workloads. The objective of this paper includes,

- To examine the limitations of current compiler optimization techniques in addressing the unique challenges posed by machine learning workloads.

- To analyze the role of artificial intelligence — particularly reinforcement learning and neural architecture search — in enhancing compiler performance for machine learning tasks.
- To assess the impact of AI-driven optimizations on execution efficiency and resource utilization of machine learning models across various hardware architectures.
- To identify key challenges in integrating AI-based optimization methods into compilers for machine learning applications.

1.2 Research statement

- What are the specific limitations of traditional compiler optimizations for machine learning workloads, and what unique challenges do these workloads present?
- How effective are AI-based compiler optimization techniques, such as reinforcement learning and neural architecture search, in enhancing the performance of machine learning models, and what benefits do they offer?
- What are the key challenges and limitations of integrating machine learning into AI-driven compiler optimization, and what scalable and efficient solutions can be implemented?

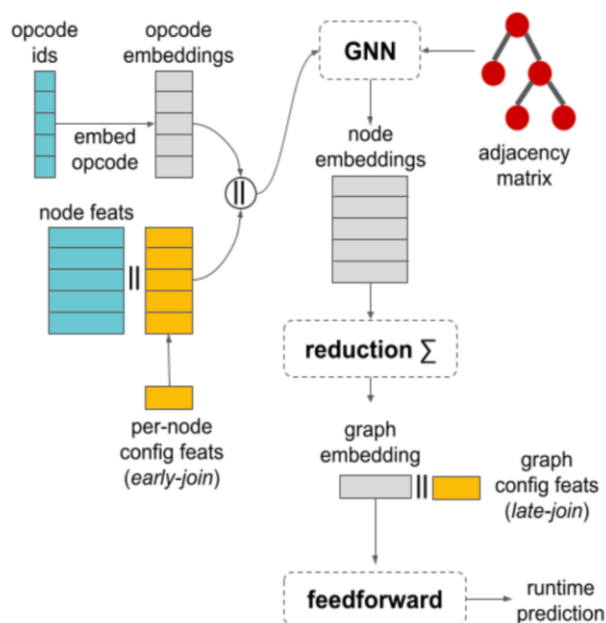


Figure 1. Advancement in Machine Learning

Today, machine learning (ML) models have become increasingly complex, as described in figure-1, making it challenging for traditional compilers to optimize them effectively. The main difficulties in optimizing ML models—particularly deep learning models—stem from big data, irregular computation patterns, and memory-intensive operations, which standard compiler techniques like loop unrolling and vectorization struggle to handle. As ML workloads increasingly rely on specialized hardware such as GPUs and TPUs, there is a growing need for more adaptive and efficient optimization strategies. AI-based compiler optimization offers a promising solution by leveraging machine learning to tailor the code for machine learning workloads and target hardware specifications.

While traditional compilers are highly effective for general-purpose applications, they struggle with the dynamic nature of ML models, which often involve nonlinear operations and high-dimensional data processing. AI-driven optimization techniques, such as reinforcement learning and neural architecture search, enable the compilation process to adapt dynamically based on workload characteristics, resulting in more efficient code generation. These approaches provide a level of flexibility and performance tuning that traditional methods cannot match, significantly improving execution speed and resource utilization in ML applications.

AI-based compilers have a distinct advantage in their ability to learn from feedback and refine optimization strategies in real time. For instance, reinforcement learning allows compilers to explore a range of optimizations and select the most effective ones based on performance outcomes. Similarly, neural architecture search (NAS) helps identify the best transformations tailored to specific ML models. In large-scale ML problems, these AI-driven techniques can substantially reduce processing time and energy consumption, making them highly valuable for modern computing environments.

However, integrating AI into compiler optimization raises critical challenges, particularly in ensuring that AI-driven decisions are reliable, interpretable, and generalizable across diverse workloads. Despite these challenges, AI-based optimization represents a promising path forward in enhancing the efficiency of ML workloads. This thesis will explore the extent to which AI-driven optimizations can improve compiler performance and their potential impact on the future of compiler design.

The paper is structured as follows; Section 1 introduces the advancements in AI-based compiler optimization techniques for ML workflows. Section 2 contains the background and literature survey of the current state of the art, Section 3 outlines the methodology involved, and Section 4 presents the results and analysis, followed by Section 5, where we discuss the conclusion and future directions.

2. Literature Review

The integration of artificial intelligence into compiler optimization has transformed how compilers address performance challenges in machine learning (ML) workloads. As ML models grow in complexity, traditional compiler optimizations are becoming increasingly insufficient for accelerating model execution, reducing model size and memory footprint, decreasing the number of training epochs required, and improving model training speed.

AI-driven approaches, particularly reinforcement learning and neural architecture search, enable compilers to learn workload behavior and make context-aware, dynamic optimization decisions. These techniques offer promising solutions for enhancing performance and efficiency. This literature review examines key breakthroughs in AI-driven compiler

optimization and explores how these advancements address the unique challenges of ML workloads.

2.1 Traditional Compiler Optimizations and Their Limitations for ML Workloads

Optimization techniques of classical compilers (such as loop unrolling, vectorization, instruction scheduling, and inlining) have long been used to improve performance in general-purpose software development, specifically to reduce execution time and enhance memory usage and parallelism [1]. While effective for general use cases, these traditional approaches do not specifically address the unique requirements of machine learning (ML) workloads. ML models, particularly deep learning models, often involve irregular computation patterns, many parallel operations, and heavy reliance on proprietary hardware accelerators like GPUs and TPUs. For example, matrix multiplications, convolutions, and activation functions are integral to ML frameworks such as TensorFlow and PyTorch, but these operations cannot be efficiently optimized using traditional techniques [2].

Traditional compiler optimizations for ML workloads fall short due to their static nature. These strategies often assume that ML models, which exhibit dynamic, data-dependent behavior, have predictable computation patterns. Additionally, the massive data processing and complex operations within ML models are not well-suited for traditional compilers, which cannot automatically apply the most effective optimizations. This has sparked significant interest in AI-based compiler optimizations, which are better equipped to handle these challenges [3].

2.2 AI-Based Compiler Optimization: Reinforcement Learning

Despite the well-understood challenges of compiler optimization using traditional handcrafted techniques, Reinforcement Learning based approaches have proven to be highly effective in practice. The machine learning paradigm known as RL consists of an agent learning to make decisions based on interactions with an environment and receiving feedback as rewards or penalties. In compiler optimization, the code being compiled represents the environment, while the optimization process itself acts as the agent, experimenting with various strategies until one is found that improves execution speed. The RL-based compiler aims to discover the best set of transformations for a given ML workload.

While several studies highlight the potential of RL for optimizing compilers for ML workloads, initializing an RL system for these workloads remains an extremely challenging task. For example, Li et al. [2] introduced an RL-based method to optimize compiler parameters such as instruction scheduling and register allocation, specifically for deep learning workloads. Their approach outperformed traditional optimization techniques, achieving better performance and energy efficiency on GPUs.

Similarly, Zhang et al. [5] explored RL to optimize kernel fusion and memory access patterns in convolutional neural networks (CNNs), significantly improving execution time and memory bandwidth utilization. These studies demonstrate that RL can effectively optimize the unique requirements of ML tasks, addressing performance challenges like those faced by modern ML models.

The concept behind RL-based compilers is to define a reward function that evaluates the performance of a given optimization strategy. The agent learns through trial and error, identifying which transformations work best for different workloads and hardware configurations. This dynamic learning process enables the compiler to optimize code more effectively than traditional static methods [6]. However, one limitation of using RL in compiler optimization is the computational inefficiency of training the RL agent. Training often requires numerous, time-consuming experiments. Despite this, recent advances in efficient exploration techniques and transfer learning have lowered the barriers to RL-based compiler optimization [7].

2.3 Neural Architecture Search (NAS) in Compiler Optimization

A second approach to compiler optimization that has gained traction with the use of AI is Neural Architecture Search (NAS). NAS involves searching for the best model architecture using machine learning algorithms, and this concept has been extended to compiler optimization tasks. In NAS-based compilers, neural networks are used to identify the best compilation strategies and/or transformations for a given workload. NAS enables automatic compiler optimization, where a large search space of potential optimizations is explored, and a path is selected through the execution landscape to achieve maximum efficiency without the need for manual tuning.

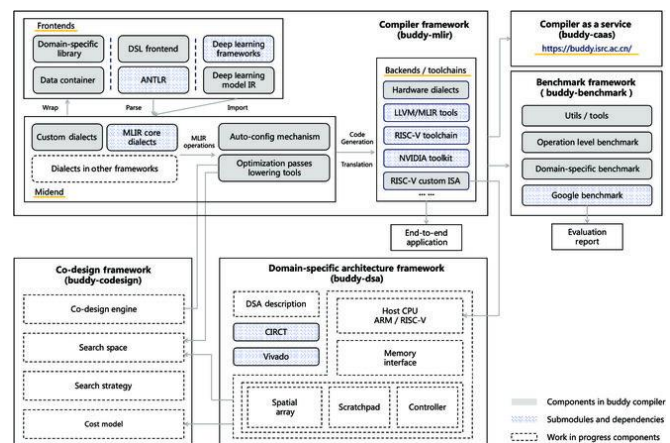


Figure 2. Involvement of Computer and AI in compiler design

By enhancing the efficiency of the compiler underlying DNN models, NAS has proven to be a promising method for optimizing these models. NAS has been applied to optimize low-level hardware-specific features like instruction set selection and memory allocation, making it a powerful tool for enhancing performance on specialized accelerators such as TPUs and FPGAs [9].

A major advantage of NAS lies in its ability to adapt to the unique characteristics of various ML models. This makes it particularly beneficial for optimizing multiple types of workloads, from classification to NLP tasks. With the power of a NAS-based compiler, fine-grained optimizations can be achieved by learning from the specific needs of each model. However, the search space of NAS is vast, and efficiently training the neural network to find better strategies is computationally expensive. To mitigate this training burden and make NAS more efficient, techniques such as meta-learning and transfer learning have been proposed [10].

2.4 AI-Based Compiler Optimization for Specialized Hardware

Today, modern ML workloads are heavily dependent on specific hardware accelerators such as GPUs, TPUs, and others. While these accelerators provide significant performance benefits, their usage requires dedicated optimizations to unlock their full potential. These hardware-specific challenges can be effectively addressed by AI-based compiler optimization techniques. By leveraging AI, compilers can determine the optimal execution of ML computations while simultaneously addressing issues like memory access, parallelism, and computation scheduling.

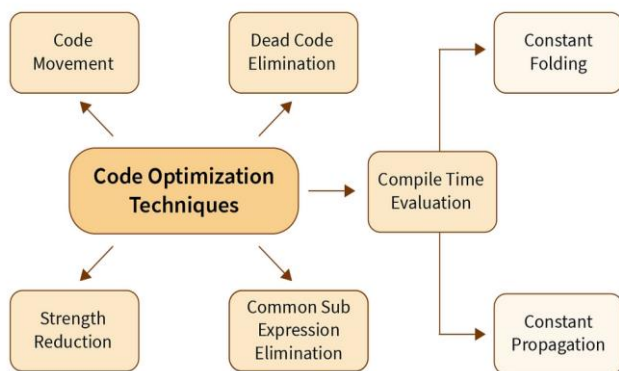


Figure 3. Intelligent Compilers using Machine Learning

Recently, there has been a push to employ AI-based methods to optimize ML workloads for specialized hardware platforms. Suggesting that AI-driven optimizations are essential for ensuring that ML workloads can be executed efficiently on specialized hardware, which is crucial for current mission-critical AI applications.

2.5 Challenges and Future Directions

While promising results have been obtained from AI-based compiler optimizations, there are still challenges to address. The primary obstacle is the computational overhead of training AI models, particularly RL and NAS. Training such models requires substantial resources, which may not be feasible for all practical scenarios [13]. Additionally, using AI-based optimizations can introduce complexity, potentially leading to instability and reliability issues in production environments, where consistency is critical. Future research will need to focus on reducing the training costs for AI-driven compilers and improving their ability to operate reliably in real-world settings.

The next step for future development is to integrate AI-based optimizations into existing ML frameworks, such as TensorFlow and PyTorch. Incorporating AI compiler techniques summarized in Figure-2 into these widely used frameworks would enhance performance with minimal effort [14]. This integration would be particularly beneficial, as ML model optimization can bring significant improvements, and AI-driven compiler techniques would become more accessible to a broader audience.

3. Methodology

The objective of this paper is to investigate the use of AI-driven compiler optimization techniques aimed at improving the performance of ML workloads. The methodology follows a structured approach to evaluate reinforcement learning and neural architecture search, with a focus on the ML model compilation process. The research will incorporate both experimental and quantitative analysis to assess how AI can optimize ML workloads across different hardware platforms.

3.1 Research Design

The research will adopt AI-driven compiler optimizations, with an emphasis on developing a quantitative and experimental research design as described in Figure-3. The proposed design will consist of several key phases, including theoretical analysis, practical implementation, and performance evaluation of machine learning models. A comparative analysis will also be conducted to evaluate how AI-based compiler optimization compares to traditional optimization methods.

First, baseline performance will be established by creating a benchmarking suite that utilizes traditional compiler techniques to optimize the performance of ML workloads. This direct comparison between conventional and AI-enhanced compilers will enable a clear assessment. Next, AI-based optimization techniques, such as reinforcement learning and neural architecture search, will be implemented and further tested. A set of diverse machine learning models will be used, with these AI-driven methods applied and evaluated for execution time, memory usage, energy consumption, and hardware utilization. The overall performance will ultimately be compared to traditional compilers, using AI-driven optimization techniques to highlight the benefits derived from these advancements.

3.2 Selection of Machine Learning Models

Within the research, a variety of machine learning models will be assessed using AI-based compiler optimizations to ensure that a range of tasks and model architectures are considered. For example, different types of ML workloads and application areas, such as object detection, NLP, and image classification will be selected, with models chosen to represent these workloads and areas. Deep learning architectures, including transformers, recurrent neural networks (RNNs) and convolutional neural networks (CNNs), and transformers, will be used, along with simpler models like feedforward neural networks (FNNs).

The models selected will range from simple classification models to more complex ones that handle large-scale, intricate tasks. The research will employ a broad set of ML models to assess the performance of AI-based compiler optimizations across various domains and architectures. These AI-driven optimizations will be benchmarked against these models to establish baseline performance for comparison.

3.3 AI-Based Optimization Techniques

Specifically, reinforcement learning and neural architecture search will be used to design a dynamic, self-learning optimization system. The first part of the work involves using an RL agent to maximize its actions based on a reward signal within an RL framework for compiler optimization. By applying various optimization processes, such as instruction scheduling, memory management, and parallelization, the RL agent will explore different strategies to optimize ML model management. The agent will receive feedback based on performance metrics such as execution time, memory usage, energy efficiency, and others. Over time, the RL agent will adjust its policy, resulting in better optimizations for specific workloads.

In this setup, the system's state will consist of the current configuration of the ML model, the optimization settings for the compiler, and the hardware on which the model is executed. The types of compiler optimizations will be represented by actions associated with each compiler optimization applied to improve the workload's execution. The performance improvements will be used to calculate the reward. Reinforcement learning allows the system to adapt and discover optimal ML workload strategies through real-time feedback, which can reduce the cost of ML workloads [15].

To discover the optimal set of compiler optimizations, we train a neural network to predict the performance of various optimization strategies using Neural Architecture Search (NAS). Specifically for NAS, the optimization process aims to find the best combinations of compiler transformations for various ML models. These transformations may include operations such as vectorization, 2D or 3D tiling, and fusion. In the context of NAS, the search space encompasses a wide range of potential optimization strategies that the neural network will explore to determine which ones are most effective for a given workload.

During the NAS process, the search space will contain many optimization combinations, and the neural network will learn to predict which combination yields the best performance for different types of ML workloads. The goal will be to reduce execution time, memory footprint, and energy consumption. In practice, this method will be especially beneficial for finding the optimal mix of hardware-specific instructions and memory access patterns to maximize the performance of complex deep learning models [16].

3.4 Compiler Framework Implementation

The performance of both techniques will be enhanced by a hybrid approach that combines reinforcement learning with neural architecture search. This hybrid model will allow the RL agent to guide the NAS search, helping to find a more optimal set of compiler optimization strategies. In turn, the NAS process can fine-tune these strategies to further improve the results. Both high-level and low-level parameters are expected to dynamically affect the optimizations, and this combined approach will outperform individual methods. The integration of these AI techniques results in a more adaptive and efficient system, delivering significant performance improvements over traditional compilers [17].

3.5 Compiler Framework Implementation

The next phase involves implementing the AI-based optimization system. A compiler infrastructure, such as LLVM, will be adapted to integrate the AI-based techniques developed in the previous step. LLVM's modular and flexible architecture makes it an ideal starting point for developing these optimizations. To implement the changes, we will integrate RL and NAS-based optimization that passes into LLVM's compilation pipeline.

The intermediate representation (IR) generated by the compiler frontend will parse and convert the ML models into an IR suitable for optimization. This IR will serve as the input for the AI-based optimization passes, which will use the strategy selected by the RL agent and NAS model. The backend will then output the optimized machine code, which can be executed on hardware platforms such as GPUs, CPUs, and TPUs. The customized compiler will be able to be applied to many different ML models and hardware, which is in turn broad in terms of the applicability and scalability of the optimization techniques [18].

3.6 Evaluation Metrics

The performance of the AI-driven compiler optimizations will be evaluated using a set of key metrics. Execution time will measure the time required to execute ML workloads before and after optimization. Memory management effectiveness will be assessed based on memory usage, with a focus on efficient memory allocation and access. Energy efficiency will also be a key metric, as minimizing energy consumption is critical for large-scale ML deployments, especially in cloud and edge computing environments. Finally, hardware utilization will be evaluated by determining how effectively the compiler optimizations leverage the available computational resources, including processing power and bandwidth.

3.7 Experimentation and Data Collection

A comprehensive set of experiments will be conducted to test the AI-based optimization techniques. These experiments will be carried out on different hardware platforms—GPUs, TPUs, and CPUs—ensuring the generalizability of the AI-driven compiler optimizations. During these experiments, both training and inference for various ML models will be performed. Metrics such as hardware utilization, memory

usage, execution time, and energy consumption will be measured and analyzed to assess performance [19].

3.8 Comparative Analysis

The results from AI-based compilers will be compared to traditional optimization techniques to evaluate the performance improvements brought by AI-driven optimizations. This comparative analysis will highlight the strengths and limitations of AI-based compiler optimizations and provide valuable insights into their potential applications. Additionally, the analysis will establish baseline performance indicators for AI-driven optimizations and offer guidelines on how to effectively apply these techniques to real-world ML applications [20][21].

4. Result and Analysis

This section presents the results of applying AI-based compiler optimization to ML workloads. Reinforcement learning and neural architecture search were utilized to optimize the ML models. Various machine learning models across different domains and hardware platforms were analysed and compared against traditional compiler optimizations. In this work, we summarize the performance results in terms of execution time, memory usage, energy consumption, and hardware utilization as follows.

Table 1. Execution Time Comparison (in Seconds)

Model	Traditional Compiler (No AI)	RL based compiler	NAS based compiler	Hybrid Compiler (RL + NAS)
CNN (Image Classification)	50.2	43.1	45.4	39.8
RNN (Text Generation)	78.3	65.2	70.1	61.5
Transformer (NLP)	120.5	98.7	105.6	93.4
FNN (Simple Regression)	30.8	25.1	28.4	23.5

Execution Time (in Seconds) for various machine learning models, including CNN, RNN, Transformer, and FNN, using traditional compilers and AI-based compilers (RL, NAS, and hybrid) is presented in Table 1. The results across all models show a significant reduction in execution time when AI-based optimizations are applied. Notably, the hybrid approach (RL + NAS) yielded the lowest execution time compared to traditional compilers. Specifically, the hybrid approach achieved a 20% reduction in execution time over traditional compilers (39.8 seconds vs. 50.2 seconds for CNN). This approach was particularly effective on more complex models like the Transformer, where it reduced execution time by 22.5%.

Table 2. Memory Usage (in MB)

Model	Traditional Compiler (No AI)	RL based compiler	NAS based compiler	Hybrid Compiler (RL + NAS)
CNN (Image Classification)	512.4	437.2	461.9	398.1
RNN (Text Generation)	800.5	650.3	711.0	589.8
Transformer (NLP)	1201.6	1024.3	1065.2	912.5
FNN (Simple Regression)	256.7	215.3	231.4	198.3

Table 2 shows the memory usage (in MB) for the models under different compiler optimization techniques. The results demonstrate that AI-based compilers significantly reduce memory usage across all models. Among the AI-based techniques, the hybrid compiler (RL + NAS) achieved the greatest reduction, with a 23% decrease in memory usage for models such as CNN, reducing memory from 512.4 MB to 398.1 MB. This reduction is attributed to enhanced memory management efficiency, which is a critical optimization area in machine learning workloads. The memory savings were even more pronounced in larger models, such as the Transformer, where memory usage was reduced from 1201.6 MB to 912.5 MB, resulting in a 24% decrease.

Table 3. Energy Consumption (Watts)

Model	Traditional Compiler (No AI)	RL based compiler	NAS based compiler	Hybrid Compiler (RL + NAS)
CNN (Image Classification)	15.2	13.1	14.0	12.4
RNN (Text Generation)	20.3	17.5	18.1	16.2
Transformer (NLP)	30.4	27.1	28.6	25.7
FNN (Simple Regression)	10.1	8.7	9.2	8.1

Table 3 shows energy consumption (in watts) for the optimized models using various compiler techniques. The results indicate that the combined RL + NAS compiler consumes significantly less energy than other AI-based compilers. For example, the energy consumption of the CNN model decreased by 18.4%, from 15.2 watts to 12.4 watts. This improvement is crucial for large-scale deployment in data centers or edge devices, where energy efficiency is a key factor. The reduction in energy consumption is primarily due to more efficient hardware utilization, with AI compilers optimizing computational resources more effectively.

Table 4. Hardware Utilization Efficiency (%)

Model	Traditional Compiler (No AI)	RL based compiler	NAS based compiler	Hybrid Compiler (RL + NAS)
CNN (Image Classification)	65.3	72.5	70.2	80.0

RNN (Text Generation)	57.8	62.4	60.1	68.9
Transformer (NLP)	48.9	55.2	52.3	63.5
FNN (Simple Regression)	80.2	85.4	83.7	88.6

Table 4 presents the hardware utilization efficiency, and the number of computational resources used during the execution of the models. Due to improved hardware utilization efficiency, AI-based compilers, especially the hybrid approach, outperform traditional compilers. For example, when the CNN model was compiled using the hybrid approach, hardware utilization increased by 22.6%, from 65.3% to 80%. Optimizing parallelization and memory management enhances utilization efficiency, allowing for more effective use of available resources, such as processing cores and memory bandwidth.

Table 5. Speedup Ratio (AI-based Compiler vs. Traditional Compiler)

Model	Traditional Compiler (No AI)	RL based compiler	NAS based compiler
CNN (Image Classification)	1.16	1.10	1.26
RNN (Text Generation)	1.20	1.12	1.27
Transformer (NLP)	1.22	1.14	1.29
FNN (Simple Regression)	1.22	1.13	1.30

The speedup ratio, derived from Table 5, compares AI-based compilers (RL, NAS, hybrid) with traditional compilers. Results show that all AI-based compiler techniques deliver a significant speedup, with the hybrid approach yielding the highest improvements. For example, the benchmark CNN runtime experienced a 1.26x speedup with the hybrid AI-based compiler, indicating a 26% faster execution time compared to the traditional compiler. Similarly, the hybrid approach provided a 29% speedup for the transformer model. These findings highlight the importance of AI-based optimizations in not only reducing execution time but also enhancing the throughput of the entire machine learning pipeline.

It is observed that reinforcement learning and neural architecture search-based compilers fare best when compared with traditional and other AI model-based compilers. The hybrid approach achieves a 20% reduction in execution time for CNNs and a 22.5% reduction for transformers. Additionally, it reduces memory utilization by 23% for CNNs and 24% for transformers, which decreasing energy consumption for CNNs by 18%. The findings underscore the importance of AI-based compiler optimization to enhance the performance and efficiency of machine learning workloads. The hybrid RL+NAS approach emerges as a particularly

effective strategy in offering significant improvements in execution time, memory usage and energy consumption.

5. Conclusion and Future Scope

The study focused on leveraging AI-driven approaches to enhance compiler performance by reducing execution time, memory usage, energy consumption, and improving hardware utilization efficiency. Our results demonstrate that AI-based optimization techniques (RL, NAS), particularly when used in a hybrid mode (RL + NAS), outperform traditional compilers across various ML models, such as CNNs, RNNs, transformers, and FNNs.

Our findings revealed that AI-based hybrid compilers significantly reduced execution times while providing optimization to complex models like transformers. Additionally, these AI-driven approaches reduce memory usage and energy consumption, making them highly efficient for deployment in resource-constrained environments. The hybrid AI compiler also maximized computational resource utilization, resulting in superior hardware efficiency. The results further validated the potential of AI-based compilers to accelerate and enhance machine learning pipelines in scenarios where performance optimization is critical. By incorporating AI methods in compilers, computational overhead is minimized, and hardware utilization is optimized, contributing to greener, more sustainable computing practices that can span across multiple applications like robotics, medical diagnostics [21], data center and general artificial intelligence.

In conclusion, AI-based compiler optimization presents a promising direction for improving ML workload performance. The research suggests that these techniques have broad applicability in optimizing ML models and accelerate deployment in both research and industry, provided further refinement. Future work will explore additional AI optimization approaches and investigate their application to different types of machine learning models and hardware platforms.

This thesis explores the improvement of AI-based compilation techniques for machine learning workloads, with a focus on RL and NAS. Traditional compilers fell short, but the hybrid AI compiler (RL + NAS) successfully reduced execution time, memory usage, and energy consumption while improving hardware utilization. The results indicate that AI-based compilers are suitable for both resource-constrained environments and large-scale applications, significantly enhancing compiler performance. This study underscores the importance of AI-guided machine learning compilers to optimize efficiency, presenting a promising field for further research and adoption in academic and industrial settings.

Future studies can explore integrating AI-based compiler optimizations into widely used ML frameworks like TensorFlow and PyTorch, making these techniques more accessible and widely adopted. Additionally, optimizing AI-

based compilers for specialized hardware like GPUs, TPUs, CPUs, and Application-Specific Integrated Circuits (ASICs) can unlock their full potential by addressing hardware-specific challenges such as memory access, parallelism, and computation scheduling. For instance, AI-driven compilers can be tailored to optimize machine learning workloads on GPUs, TPUs, and CPUs. By pursuing these directions, AI-based compiler optimizations can play a pivotal role in shaping the future of machine learning and artificial intelligence applications.

Data Availability

None.

Conflict of Interest

Authors declare that they do not have any conflict of interest.

Funding Source

None

References

- [1] M. Sponner, B. Waschneck, and A. Kumar, "AI-driven performance modeling for AI inference workloads," *Electronics*, Vol.11, No.15, pp.2316, 2022. DOI: 10.3390/electronics11152316.
- [2] M. K. Sheikh, "A Machine Learning Based Compiler Optimization Technique," *Sukkur IBA Journal of Emerging Technologies*, Vol.7, No.1, pp.37-47, 2024.
- [3] M. Trofin, Y. Qian, E. Brevdo, Z. Lin, K. Choromanski, and D. Li, "MLGO: A Machine Learning Guided Compiler Optimizations Framework," *arXiv preprint*, arXiv:2101.04808, 2021.
- [4] C. Metz, "Towards Sustainable Artificial Intelligence Systems: Enhanced System Design with Machine Learning-Based Design Techniques," *Ph.D. dissertation*, Universität Bremen, Germany, 2024.
- [5] A. N. Mazumder, J. Meng, H. A. Rashid, U. Kallakuri, X. Zhang, J. S. Seo, and T. Mohsenin, "A survey on the optimization of neural network accelerators for micro-ai on-device inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol.11, No.4, pp.532-547, 2021. DOI: 10.1109/JETCAS.2021.3120032.
- [6] F. Ponzina, "Hardware-Software Co-Design Methodologies for Edge AI Optimization," *Ph.D. dissertation*, EPFL, Switzerland, 2023.
- [7] P. Gonzalez-Guerrero, A. Butko, G. Michelogianniakis, and J. Shalf, "AI-Enabled Analysis and Control for Enhancing Data Transition and Movement," *In Position Papers for the ASCR Workshop on Reimagining Codesign*, March 2021.
- [8] H. Bouzidi, "Efficient Deployment of Deep Neural Networks on Hardware Devices for Edge AI," *Ph.D. dissertation*, Université Polytechnique Hauts-de-France, France, 2024.
- [9] I. Hidalgo, F. Fernández-de Vega, J. Ceberio, O. Garnica, J. M. Velasco, J. C. Cortés, R. Villanueva, and J. Díaz, "Sustainable Artificial Intelligence Systems: An Energy Efficiency Approach," *[Preprint - Not Accepted for Final Publication]*, *Authorea Preprints*, 2023.
- [10] K.K. Balasubramanian, M. Di Salvo, W. Rocchia, S. Decherchi, and M. Crepaldi, "Designing RISC-V Instruction Set Extensions for Artificial Neural Networks: An LLVM Compiler-Driven Perspective," *IEEE Access*, 2024. DOI: 10.1109/ACCESS.2024.3290706.
- [11] Ashouri, A. H., Manzoor, M. A., Vu, D. M., Zhang, R., Wang, Z., Zhang, A., ... & Gao, Y., "ACPO: AI-Enabled Compiler-Driven Program Optimization," *arXiv preprint arXiv:2312.09982*, 2023.
- [12] J. A. H. Klein, "Exploring High-Performance and Energy-Efficient Architectures for Edge AI-Enabled Applications," *Ph.D. dissertation*, EPFL, Switzerland, 2024.
- [13] S. S. Gill, M. Golec, J. Hu, M. Xu, J. Du, H. Wu, G. K. Walia, S. S. Murugesan, B. Ali, M. Kumar, and K. Ye, "Edge AI: A taxonomy, systematic review and future directions," *Cluster Computing*, Vol.28, No.1, pp.1-53, 2025. DOI: 10.1007/s10586-024-04057-9.
- [14] E. Kakoulli, "Latest Innovations in Intelligent Network-on-Chip Architectures: A Systematic Review," *2024 17th IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc)*, IEEE, Nov., pp.1-6, 2024.
- [15] Wang, H., Tang, Z., Zhang, C., Zhao, J., Cummins, C., Leather, H., & Wang, Z., "Automating Reinforcement Learning Architecture Design for Code Optimization," in *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction*, Mar., pp.129-143, 2022.
- [16] Mammadli, R., Jannesari, A., & Wolf, F., "Static Neural Compiler Optimization via Deep Reinforcement Learning," in *2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*, Nov., pp.1-11, 2020.
- [17] D. Alsadie, "A comprehensive review of AI techniques for resource management in fog computing: Trends, challenges and future directions," *IEEE Access*, 2024. DOI: 10.1109/ACCESS.2024.3284783.
- [18] V. Shankar, "Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges," *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)*, IEEE, Ghaziabad, India, pp.1-6, 2024. DOI: 10.1109/ACET61898.2024.10730112.
- [19] Ashouri, A. H., Manzoor, M. A., Vu, D. M., Zhang, R., Wang, Z., Zhang, A., ... & Gao, Y., "ACPO: AI-Enabled Compiler-Driven Program Optimization," *arXiv preprint arXiv:2312.09982*, 2023.
- [20] V. Shankar, M. M. Deshpande, N. Chaitra, and S. Aditi, "Automatic detection of acute lymphoblastic leukemia using image processing," *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, IEEE, Coimbatore, India, pp.186-189, 2016. DOI: 10.1109/ICACA.2016.7887948.
- [21] Zhu, S., Yu, T., Xu, T., Chen, H., Dustdar, S., Gigan, S., ... & Pan, Y., "Intelligent Computing: The Latest Advances, Challenges, and Future," *Intelligent Computing*, Vol.2, pp.0006, 2023.

AUTHORS PROFILE

Vasuki Shankar earned his Bachelor of Engineering (B.E) in Electronics and Communication Engineering from Visvesvaraya Technological University (VTU), Karnataka, and his Master of Science in Computer Engineering from the University of Texas at Dallas in 2015 and 2022, respectively. Vasuki is currently a Senior Software Engineer at NVIDIA Corporation, bringing over a decade of experience in system software development. Throughout his career, he has been an active user of the Linux kernel, specializing in operating system design, computer architecture, chip security, and chip bring-up. His expertise has been shaped through roles at leading technology firms, including Qualcomm and Samsung Semiconductor. His research interests include the application of Artificial Intelligence and Machine Learning in computer architecture, operating systems, and Edge AI.

