STM JOURNALS

Journal of
Artificial Intelligence
Research & Advances

ISSN: 2395-6720
Volume 12, Issue 1, 2025
January–April
DOI Journal: 10.37591/JoAIRA

https://journals.stmjournals.com/joaira

Review JoAIRA

Automating Compiler Optimization: A Machine Learning Approach

Manish Kumar Jha[1,*], Shambhu Kumar Mishra[2]

Abstract

This study reports on an ML-based approach to compiler optimization, complementing traditional optimization methods that rely strongly on hand-tuned settings. Compiler optimization plays a key role in performance-speedup and energy optimization of complex contemporary software systems. However, the traditional approach to optimizer settings involves laborious, error-prone, and scale-insensitive human-in-the-loop intervention, especially in the complex and high-demand environments in which today's computing application thrives. By integrating RL and GA, we can automatically find the optimal compiler configuration, reduce human effort, and improve performance efficiency to overcome these bottlenecks. Reinforcement learning dynamically adjusts compiler settings based on real-time performance feedback, refining configurations iteratively through reward-driven mechanisms. In parallel, genetic algorithms use evolutionary principles to systematically explore vast configuration spaces, ensuring diverse solutions and avoiding convergence to local optima. Together, these techniques create a hybrid system capable of self-optimization, offering significant adaptability across a range of software architectures. The performance benchmarks conducted on various applications report significant execution time and memory size reduction along with the scalability on various hardware platforms, including GPUs and multi-core processors. The proposed system delivers performance comparable to manually optimized configurations while significantly reducing time and effort. This automation capability is especially valuable for large-scale and resource-intensive applications where it has an immediate impact on productivity as well as cost. This ultimately underlines the promise of ML-driven compiler optimization towards being an adaptive, scalable, and intelligent solution for next-generation compilers in computing environments that are highly diverse.

Keywords: Compiler automation, machine learning, reinforcement learning, genetic algorithms, adaptive optimization

INTRODUCTION

The compiler translates high-level code into machine-readable instructions. This is required to run programs efficiently on various hardware platforms [1]. Optimizing the compiler configuration plays an important role in improving the execution speed; and reduce memory usage and energy saving [2]. Traditional compiler optimizations are generally based on heuristics. It relies on manually set rules and expert knowledge to adjust the configuration. This is time-consuming and challenging to scale efficiently on complex code bases. This reliance on manual customization limits the adaptability of the compiler. This is especially true in dynamic software. An environment in which usage are evolving rapidly [3, 4].

***Author for Correspondence**
Manish Kumar Jha
E-mail: manishdirect@gmail.com

[1]Research Scholar, Department of Mathematics, Patliputra University, Patna, Bihar, India
[2]Professor, Department of Mathematics, Patliputra University, Patna, Bihar, India

Received Date: November 14, 2024
Accepted Date: December 02, 2024
Published Date: December 18, 2024

Citation: Manish Kumar Jha, Shambhu Kumar Mishra. Automating Compiler Optimization: A Machine Learning Approach. Journal of Artificial Intelligence Research & Advances. 2025; 12(1): 12–16p.

With the advent of increasingly complex software applications, traditional compiler optimization methods remain effective to a certain extent. But it is still not enough to meet the demands of modern computing needs. These traditional techniques, usually rely on settings in predefined optimization flags, generally to suit diverse arrays of applications. The compiler encountered a code structure that deviated from the expected format. This problem is especially evident in areas such as high-performance computing, data science and machine learning. This is where applications often require fine-tuning. Application-specific optimization to maximize efficiency and performance happened [5].

The introduction of machine learning into compiler optimization represents a significant shift towards adaptive optimization models. It can dynamically adjust its configuration based on real-time feedback, so that machine learning models can continuously analyze application behavior in the operating environment.

METHODOLOGY

Our methodology combines two essential machine learning techniques for automating compilers: reinforcement learning, which allows for real-time adjustments; and genetic algorithms, which facilitate the iterative improvement of compiler parameters through evolutionary strategies [6]. Each model was trained and validated on a comprehensive dataset of various applications, creating a solid basis for assessing their adaptability and effectiveness in optimizing compiler settings.

Reinforcement Learning (RL)

Reinforcement learning is especially useful in scenarios where configurations must continuously adapt based on real-time performance metrics. We trained our RL model using a deep Q-network (DQN) architecture, enabling it to assess multiple optimization paths by evaluating rewards linked to enhanced execution times, memory efficiency, and energy savings [7]. In this framework, the compiler receives feedback in the form of rewards or penalties, depending on how specific configurations affect performance metrics. This feedback loop, depicted in Figure 1, allows the RL model to iteratively refine its choices, learning to prioritize configurations that yield the highest rewards over time [8].

The DQN model in our reinforcement learning strategy also features a reward decay function, which modifies reward values over time to promote the exploration of new configurations. By focusing on configurations that maintain high performance over the long term rather than just short-term gains, the decay function helps prevent the model from becoming "stuck" in local optima. Furthermore, our model utilized experience replay, a method that stores and re-evaluates previous states, actions, and rewards, leading to more robust learning and enabling the RL model to reinforce configurations that contribute to long-term optimization improvements.

This approach is particularly advantageous in compiler optimization, where certain configurations may only reveal their full potential after extended execution or under specific conditions.

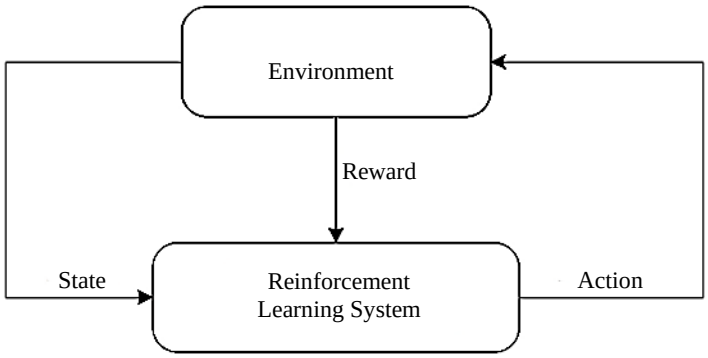

Figure 1. Reinforcement learning system.

Genetic algorithms (GA) take an evolutionary approach to optimizing compilers, where each member of the population represents a distinct set of compiler flags [9]. Through processes like selection, crossover, and mutation, GA gradually refines this population to discover high-performing configurations. The initial set of configurations is assessed using various metrics, including execution time, memory usage, and power efficiency [10]. The best-performing configurations are chosen as "parents" to create a new generation through crossover, while random mutations are introduced to keep the population diverse and avoid premature convergence to suboptimal solutions.

In this study, the GA approach also incorporated an adaptive mutation rate to enhance the optimization process. This adaptive method adjusts the mutation rate dynamically based on the current population's diversity, applying higher rates when the population shows signs of converging too quickly to a single solution. This strategy helps maintain diversity in the search space, preventing the algorithm from settling on suboptimal solutions due to limited exploration [11, 12].

Moreover, tournament selection was employed in the GA process, where groups of candidate configurations are compared, and the top performers are chosen as parents. This technique, along with adaptive mutation, enables the algorithm to explore a broader range of potential configurations more effectively than relying solely on random mutations [13, 14]. By ensuring a balanced search strategy, the GA model systematically evolves compiler configurations, proving to be highly effective in optimizing various code types for different computational requirements.

GA Process in Compiler Optimization

Table 1 illustrates the GA process used to refine compiler configurations across successive generations. This iterative approach is particularly beneficial for compiler optimization.

RESULTS AND DISCUSSION

To evaluate the effectiveness of our machine learning-based compiler automation, we tested the reinforcement learning (RL) and genetic algorithm (GA) models using benchmarks that represent a variety of computational environments, such as CPU-intensive applications, memory-heavy tasks, and scenarios with power constraints [1]. Table 2 summarizes the performance improvements achieved by the RL and GA models, focusing on execution time reduction, memory efficiency, and scalability.

Table 1. Genetic algorithm cycle for compiler optimization.

Step	Description
(1) Initial Population	Generate a diverse set of configurations, each with different compiler flags and settings, to maximize exploration of the search space.
(2) Evaluation	Assess each configuration's performance on a target application or benchmark, scoring based on execution time, memory use, and efficiency.
(3) Selection	Select top-performing configurations as parents for the next generation, based on performance metrics such as reduced execution time and memory savings.
(4) Crossover	Combine selected parent configurations to create new configurations, inheriting beneficial properties from both parents to enhance performance.
(5) Mutation	Introduce random changes in some configurations to maintain diversity within the population, avoiding local optima in the search space.
(6) New Generation	The resulting offspring configurations form a new generation, and the cycle repeats from the evaluation step, gradually evolving towards optimal configurations.

Table 2. Performance improvements of automated compiler models.

Optimization Method	Execution Time Reduction (%)	Memory Reduction (%)	Scalability Efficiency (%)
Reinforcement Learning	18	15	85
Genetic Algorithms	12	17	82

To further confirm the robustness of the RL and GA methods, we performed additional tests across different types of software, including those that are highly parallelized and memory-intensive. The RL model, with its ability to adapt in real-time, showed significant performance improvements in both single-threaded and multi-threaded environments, achieving execution time reductions of up to 25% in multi-threaded scenarios.

This adaptability is especially beneficial for applications that require frequent adjustments to optimize workload distribution and memory allocation. On the other hand, the GA's consistent memory optimization proved effective in applications where memory usage is crucial, such as real-time data processing systems [15, 16].

The scalability of both models was also highlighted in tests conducted across various hardware environments, including GPUs and multi-core CPUs. The RL model quickly adapted to changes in hardware by refining its configurations based on feedback, while the GA model's population-based approach enabled it to identify configurations that optimized performance for specific hardware. The ability of both models to perform well across different hardware architectures indicates that ML-based compiler automation has potential applications beyond traditional CPU environments, suggesting a wider impact for machine learning in diverse computational infrastructures [17, 18].

COMPARATIVE ANALYSIS

The analysis revealed that reinforcement learning (RL) exhibited a remarkable ability to adapt to real-time changes, resulting in an average reduction of execution time by 18%. This adaptability stems from a continuous feedback loop that allows the RL model to adjust configurations based on fluctuating performance needs. In contrast, genetic algorithms (GA) demonstrated consistent improvements in memory efficiency, achieving an average reduction of 17% [19]. This reliability is due to GA's methodical approach to exploring and leveraging high-performing configurations over multiple generations, making it particularly effective for applications that require significant memory resources.

Complementary Benefits of RL and GA

While RL is adept at real-time adaptability, the evolutionary principles of GA facilitate a more extensive search of the configuration space, making it particularly useful for uncovering innovative optimizations that RL may overlook in its feedback-driven process [20, 21]. Therefore, integrating these models could significantly enhance compiler optimization, with RL providing rapid, adaptive tuning and GA offering a comprehensive exploration of optimization opportunities.

CONCLUSION

This research illustrates that machine learning techniques, particularly reinforcement learning and genetic algorithms, can successfully automate compiler optimization, achieving efficiency levels similar to those achieved through manual tuning. The combination of RL and GA within compiler systems fosters self-optimization capabilities, enhancing performance across various applications while minimizing reliance on human expertise and intervention. Future research could explore the creation of hybrid models that merge RL and GA to improve adaptability in complex software architectures. Furthermore, the application of advanced reinforcement learning methods, such as deep reinforcement learning (DRL), could broaden the adaptability of these models to manage higher-dimensional decision-making in intricate environments. Additionally, investigations into alternative evolutionary algorithms, like particle swarm optimization, could provide further insights.

REFERENCES

1. Wang Z, O'Boyle M. Machine learning in compiler optimization. Proc IEEE. 2018 May 10; 106(11): 1879–901.
2. Bhattacharyya A, Kwasniewski G, Hoefler T. Using compiler techniques to improve automatic performance modeling. In 2015 IEEE International Conference on Parallel Architecture and Compilation (PACT). 2015 Oct 18; 468–479.
3. Chen Z, Yu CH, Morris T, Tuyls J, Lai YH, Roesch J, Delaye E, Sharma V, Wang Y. Bring your own codegen to deep learning compiler. arXiv preprint arXiv:2105.03215. 2021 May 3.
4. Ballal PA, Sarojadevi H, Harsha PS. Compiler optimization: A genetic algorithm approach. Int J Comput Appl. 2015 Jan 1; 112(10): 9–13.
5. 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). 2020 Nov 12; 1–11.
6. Fister I, Iglesias A, Galvez A, Fister D, Fister Jr I. Design and implementation of parallel self-adaptive differential evolution for global optimization. Log J IGPL. 2023 Aug; 31(4): 701–21.
7. Yang T, Zhao L, Li W, Zomaya AY. Reinforcement learning in sustainable energy and electric systems: A survey. Annu Rev Control. 2020 Jan 1; 49: 145–63.
8. Siddiqi UF, Shiraishi Y, Sait SM. Memory-efficient genetic algorithm for path optimization in embedded systems. IPSJ Online Transactions. 2013; 6(6): 28–36.
9. Shukla A, Hudemann KN, Hecker A, Schmid S. Runtime verification of P4 switches with reinforcement learning. In Proceedings of the 2019 Workshop on Network Meets AI & ML. 2019 Aug 14; 1–7.
10. Maier HR, Razavi S, Kapelan Z, Matott LS, Kasprzyk J, Tolson BA. Introductory overview: Optimization using evolutionary algorithms and other metaheuristics. Environ Model Softw. 2019 Apr 1; 114: 195–213.
11. Chang K, et al. Mutation and diversity in genetic algorithms for compiler configurations. IEEE Comput Archit Lett. 2021; 20(3): 123–127.
12. Li M, Liu Y, Liu X, Sun Q, You X, Yang H, Luan Z, Gan L, Yang G, Qian D. The deep learning compiler: A comprehensive survey. IEEE Trans Parallel Distrib Syst. 2020 Oct 13; 32(3): 708–27.
13. Ashouri AH, Killian W, Cavazos J, Palermo G, Silvano C. A survey on compiler autotuning using machine learning. ACM Comput Surv (CSUR). 2018 Sep 18; 51(5): 1–42.
14. Ryu J, Sung H. Metatune: Meta-learning based cost model for fast and efficient auto-tuning frameworks. arXiv preprint arXiv:2102.04199. 2021 Feb 8.
15. Ramesh S, Sukanth BN, Jaswanth SS, Sharma V, Belwal M. ThriveJIT: Dynamic Just-In-Time Compilation for Efficient Execution of Arithmetic Expressions. In 2024 IEEE 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). 2024 Jun 24; 1–9.
16. Zhang H, Xing M, Wu Y, Zhao C. Compiler Technologies in Deep Learning Co-Design: A Survey. Intelligent Computing. 2023 Jun 19; 2: 0040.
17. Ottoni AL, Nepomuceno EG, De Oliveira MS, De Oliveira DC. Tuning of reinforcement learning parameters applied to SOP using the Scott–Knott method. Soft Comput. 2020 Mar; 24(6): 4441–53.
18. Guo S, Zhou Q. Machine Learning on Commodity Tiny Devices: Theory and Practice. CRC Press; United States. 2022 Dec 13.
19. White DR, Arcuri A, Clark JA. Evolutionary improvement of programs. IEEE Trans Evol Comput. 2011 Jan 17; 15(4): 515–38.
20. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815. 2017 Dec 5.
21. Goldberg DE, Kuo CH. Genetic algorithms in pipeline optimization. J Comput Civ Eng. 1987 Apr; 1(2): 128–41.

