



Applying Machine Learning to Optimize Software Runtime and Memory Usage

Alireza Rahimipour Anaraki*, Parvaneh Asghari.

1. Dep. Computer Engineering, CT.C., IAU., Tehran, Iran, a.rahimipouranaraki@iau.ir
2. Dep. of Computer Engineering, CT.C, IAU., Tehran, Iran, p_asghari@iauctb.ac.ir

Abstract

Software systems today face increasing demands for high performance and efficient memory usage. Traditional optimization methods, often hard-coded and inflexible, struggle to adapt to complex and dynamic workloads. Machine Learning (ML) algorithms offer a promising approach by automatically learning patterns and making intelligent decisions to optimize execution time (runtime) and memory consumption. This paper provides a comprehensive review of recent research (2021–2025) on applying ML techniques – including neural networks, reinforcement learning, and evolutionary algorithms – to performance optimization in software systems. We discuss how ML-driven solutions have achieved significant improvements, such as reducing program execution time, enhancing memory/cache efficiency, and intelligently allocating resources in cloud environments.

Key words: Machine Learning; Performance Optimization; Execution Time; Memory Usage; Evolutionary Algorithms; Cloud Computing.



1. Introduction

Improving the runtime performance and memory efficiency of software systems is a longstanding challenge in computer science. Execution speed (low latency, fast completion) and memory usage (efficient allocation, caching, minimal waste) directly impact system throughput, user experience, and operational costs. Traditional optimization techniques in software – such as fixed heuristics in compilers, operating systems, or databases – are typically designed for specific scenarios and lack adaptability to changing workloads. As modern software operates in increasingly dynamic environments (e.g., cloud data centers, mobile devices) with varying input patterns, these static approaches often fail to meet complex performance requirements (Sanjalawe et al., 2025). This gap has motivated the exploration of machine learning-based solutions that can learn and adapt to optimize system performance continuously.

Machine learning (ML) algorithms can automatically discover complex patterns in data and make predictions or decisions, which is a key advantage for system optimization. Rather than relying on a human's manual tuning or intuition, ML-based methods learn from data how to improve performance. For example, ML models can identify non-linear relationships in program behavior that traditional methods miss; as noted by Choi and Park (2021), “machine learning can find the various relationship among data... and automatically explore patterns for a given workload” (Choi & Park, 2021). By leveraging techniques such as neural networks, reinforcement learning, and evolutionary algorithms, researchers have begun to design self-optimizing systems that adjust their behavior for better speed and efficiency.

Recent studies show the promise of ML in system performance tuning. Choi and Park (2021) survey several ML-based optimization approaches across system components (e.g. CPU branch prediction, caching, data structures), and conclude that this emerging field has “significant potential for future research” (Choi & Park, 2021). In practice, machine learning has already been applied to tasks like database index selection, memory cache management, and code optimization with notable success. For instance, preliminary results demonstrate that incorporating learning models can outperform hand-crafted policies, especially under unpredictable workloads (Choi & Park, 2021). Moreover, industry leaders have started integrating ML into cloud platforms for resource management, validating its practical viability (Anbarkhan, 2025).

2. Research structure

The remainder of this paper is organized as follows. Section 1 provides background on machine learning techniques for runtime and memory optimization, highlighting how



different ML algorithms contribute to performance improvements. Section 2 focuses on evolutionary algorithms as a subset of ML techniques, reviewing their application in optimizing execution time and memory usage. Section 3 discusses the use of ML in dynamic and cloud-based software systems, where adaptability is crucial. Section 4 analyzes key challenges and future directions for ML-driven optimization in software systems. Finally, Section 5 presents the conclusions of this review.

1 .Machine Learning Techniques for Performance Optimization

1.1 Adaptive Performance Tuning with ML:

Machine learning-based optimization differs fundamentally from traditional fixed heuristics by adapting to data. An ML model can be trained on performance data (e.g., past executions, memory access patterns) and subsequently make decisions that improve speed or efficiency. This adaptivity is crucial for modern systems where workload characteristics can change over time. Traditional methods like static caching policies or manual code optimizations cannot easily adjust once deployed, leading to suboptimal performance if conditions deviate from the original assumptions. In contrast, ML models (such as neural networks or decision trees) can generalize from training data and even continue learning online to respond to new patterns. According to Choi and Park (2021), a machine learning model can capture various data distributions, whereas a traditional model typically works well only for the specific cases it was designed for (Choi & Park, 2021). This means ML approaches can learn complex behaviors (e.g., irregular memory access patterns or workload spikes) that are hard to handle with static rules.

1.2 Supervised Learning for Prediction and Optimization:

A common use of ML in system optimization is to predict future behavior and adjust accordingly. Supervised learning models are trained on historical data to predict metrics like execution time, memory usage, or resource demand. For example, perceptron or multi-layer neural networks have been used to replace parts of system algorithms: a neural model can predict the location of data in an index or the outcome of a branch, improving upon static structures. In database index management, learned models predict key positions more accurately than fixed B-tree indexes, speeding up queries (Choi & Park, 2021). Similarly, neural networks have been embedded in cache prefetching systems to predict which memory blocks will be reused, thus reducing cache misses. One study introduced a perceptron-based cache prefetcher that learns to reject unnecessary prefetch requests, significantly cutting down wasted memory



bandwidth (Choi & Park, 2021). By analyzing access patterns, the perceptron could decide not to fetch data that it predicts will not be used, leading to higher cache hit rates. This illustrates how learning-based components within traditional algorithms can optimize performance metrics beyond what fixed strategies achieve.

Another example is ML-augmented branch prediction in CPUs. Traditional branch predictors use heuristic tables and finite state machines, which have limited capacity and cannot adapt to every program's branching pattern. A learned branch predictor uses a neural network (perceptron or even LSTM) to predict branch directions from historical execution data. Research has shown these learned predictors can improve accuracy, thereby reducing misprediction penalties and improving runtime. In summary, supervised ML models in system components serve to predict outcomes (hits, misses, branch taken or not, etc.) more intelligently and thus optimize the execution flow and resource usage.

1.3 Reinforcement Learning for Dynamic Optimization:

Reinforcement learning (RL) is another powerful approach, particularly suited for dynamic decision-making in systems. In an RL framework, an agent (the system or component) learns a policy by trial-and-error to maximize a reward (e.g., throughput, cache hits, or energy efficiency). RL has been effectively applied to problems like CPU resource scheduling, memory management, and cloud autoscaling. For instance, cache replacement algorithms enhanced with reinforcement learning can learn which pages to evict by receiving feedback on whether a choice led to a cache miss or hit later (Shankar et al., 2025). Instead of using a fixed policy like Least Recently Used (LRU), an RL-based page replacement can adapt to access patterns, resulting in fewer misses. Shankar et al. (2025) highlight that such intelligent page replacement via reinforcement learning can outperform traditional policies by utilizing real-time feedback to make better decisions (Shankar et al., 2025).

In cloud computing, reinforcement learning agents have been used to auto-scale resources (CPU, memory, etc.) in response to changing demand. A notable example is Google Cloud Platform's use of an RL-based system that continuously learns and adjusts resource allocation in real-time (Anbarkhan, 2025). The RL agent observes the system state (load, response times) and takes actions (provision or de-provision resources) to maximize performance and minimize cost. Over time, it learns an effective scaling policy that traditional threshold-based autoscalers cannot match. This adaptive approach is crucial for cloud services where workload can be "erratically variable" and hard-coded rules either over-provision (wasting resources) or under-

provision (causing slowdowns). By predicting and reacting to workload changes, ML-driven resource management achieves a better balance.

1.4 Memory Optimization with AI:

Memory systems (caches, RAM, etc.) benefit greatly from ML due to their complex usage patterns. AI techniques like deep learning have been employed for predictive caching, where models anticipate which data will be needed and pre-load it into faster memory. This reduces wait times for data fetch and improves runtime performance. Additionally, dynamic memory allocation has been enhanced by AI – instead of static allocators, AI-driven allocators adjust memory distribution based on the program’s behavior (Shankar et al., 2025). These approaches collectively lead to more efficient memory utilization. A recent study by Shankar et al. (2025) demonstrated remarkable gains: AI-based memory optimizations yielded up to a 40% reduction in memory access latency and a 15–20% increase in cache hit rates in modern multicore processors (Shankar et al., 2025). This was achieved through techniques such as predictive prefetching and smarter replacement policies that learn from past usage. The same study noted that these ML techniques also lowered energy consumption, since fewer slow memory accesses (which consume more power) were needed. Figure 1 below illustrates an example of ML-driven improvements in resource utilization metrics, comparing baseline and ML-optimized scenarios.

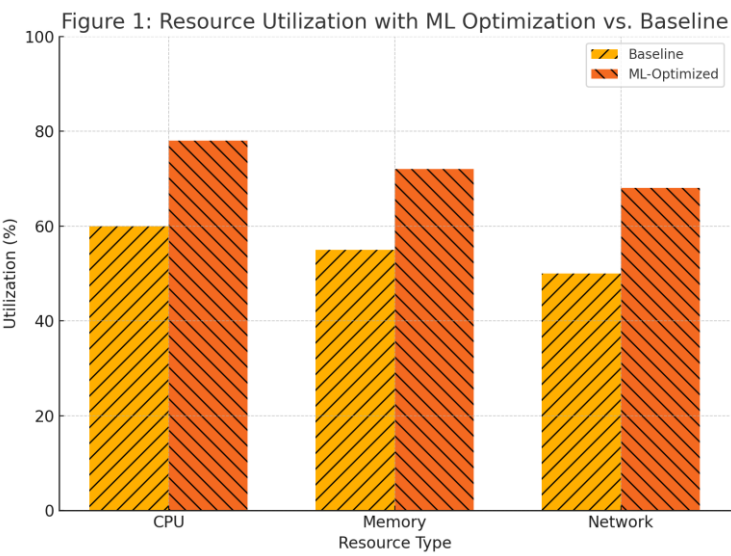


Figure 1: Resource Utilization With ML Optimization Vs. Baseline

As Figure 1 suggests, ML-based systems can drive resources closer to optimal use without human intervention. The dynamic adjustments ensure that computing



resources (CPU cycles, memory space, bandwidth) are not left idle unnecessarily, nor overwhelmed due to slow reaction. The outcome is improved execution times (since resources are available when needed) and better memory usage (since memory is allocated or freed intelligently). In summary, machine learning techniques enable continuous, data-driven optimization of performance: they predict future needs, preemptively adjust systems, and learn from results to refine their strategies.

1.5 Benefits and Limitations:

The advantages of ML in performance tuning are clear in terms of adaptivity and potential gains. Systems enhanced with learning can handle complex patterns and non-linear relationships in workload behavior that manual methods often cannot (Choi & Park, 2021). However, it is important to acknowledge the limitations and overhead that ML introduces. Training an ML model requires data – sometimes large amounts – and this training can be time-consuming. Models like deep neural networks might also incur extra computation at runtime (for making predictions or decisions), which itself consumes CPU and memory. Choi and Park (2021) point out that ML models depend on the quality of their training dataset and typically need time to generate or update the model, which could be seen as a disadvantage relative to traditional hard-coded methods (Choi & Park, 2021). There is a trade-off between the overhead of learning and the performance benefit gained. In many cases, once an ML model is trained and if it generalizes well, the runtime overhead is negligible compared to the performance improvement it provides. But in scenarios where training data is scarce or the environment changes faster than the model can adapt, the benefits might be limited. Despite these caveats, recent research overwhelmingly suggests that with careful design (and sometimes using lightweight or specialized models), the net effect of ML is positive for optimizing runtime and memory usage (Choi & Park, 2021; Shankar et al., 2025). Table 1 summarizes some real examples of ML applications in this domain and the improvements achieved.

Table 1: Examples of ML Applications in Software Performance Optimization (2021–2025). Each example highlights the approach used and the resulting performance gains. (All improvements are compared to a baseline without the ML technique.)

Application Area	ML Technique	Reported Improvement
Cloud Resource Allocation (Anbarkhan, 2025)	Workload prediction (regression & neural network ensemble)	+30% resource utilization, –25% cost (dynamic scaling reduced idle resources and met demand)
Memory & Cache Management (Shankar et al., 2025)	Reinforcement Learning for caching; Predictive analytics for prefetch	–40% memory access latency; +15–20% cache hit rate (fewer cache misses, faster memory access)
GPU Kernel Code Optimization (Liou et al., 2024)	Evolutionary algorithm (genetic programming of code, “GEVO”)	–17% to –29% execution time for various applications (auto-tuned code outperformed manual optimizations)
Real-Time Task Scheduling (Hassan et al., 2025)	Genetic Algorithm for scheduling on multiprocessors	Eliminated missed deadlines (0 misses) and minimized average response time under high load (improved timeliness and throughput)

As shown in Table 1, ML techniques span a broad range of applications – from low-level code generation to high-level cloud management – and in each case they delivered notable performance and efficiency gains. These examples also indicate that different types of ML (supervised learning, reinforcement learning, evolutionary algorithms) can be tailored to specific optimization problems. In the following sections, we delve deeper into evolutionary algorithms (a subset of machine learning/AI techniques) and the context of cloud and dynamic systems, which are particularly prominent areas for these advancements.

2. Evolutionary Algorithms for Runtime and Memory Optimization

Evolutionary algorithms are a category of optimization techniques inspired by natural evolution (genetic selection and mutation). They fall under the broader umbrella of artificial intelligence and are often considered alongside machine learning approaches,

especially for search and optimization problems. Genetic Algorithms (GA), Genetic Programming (GP), and other evolutionary strategies iteratively improve a population of candidate solutions for a given problem, guided by a fitness function. In the context of software performance, the "solutions" being evolved could be scheduling sequences, configuration parameters, or even segments of code, and the fitness function is defined to reward faster execution time, lower memory usage, or other performance metrics.

2.1 Role of Evolutionary Algorithms:

Unlike predictive ML models, evolutionary approaches do not require a training dataset of input-output pairs; instead, they explore the space of possible solutions through mechanisms of crossover (recombining parts of solutions), mutation (randomly tweaking a solution), and selection (choosing the better solutions to form the next generation). This makes them well-suited for complex optimization landscapes where the exact relationship between design decisions and performance outcome is not fully understood or is highly non-linear. Evolutionary algorithms have been applied to scenarios such as multi-objective optimization, where trade-offs exist between, say, execution speed and memory consumption or energy usage. They can search for solutions that balance these objectives. For example, Sanjalawe et al. (2025) note that meta-heuristic techniques like genetic algorithms or Particle Swarm Optimization can effectively "balance trade-offs between energy consumption, execution time, and cost" in scheduling problems by exploring a wide solution space (Sanjalawe et al., 2025). This ability to navigate complex trade-off surfaces is a strength of evolutionary methods in performance tuning.

2.2 Genetic Algorithms in Scheduling and Allocation:

A prominent use case for GAs is in task scheduling on processors or in cloud environments. Scheduling is essentially a combinatorial optimization problem – finding an ordering or assignment of tasks to resources that optimizes criteria like total execution time (makespan), resource utilization, or deadline adherence. Genetic algorithms have been successfully used to evolve near-optimal schedules. Hassan et al. (2025) present an innovative GA-based scheduler for real-time multiprocessor systems. Their approach evolves task orderings to minimize schedule length and meet all deadlines. The results were impressive: the GA scheduler "consistently achieved zero missed deadlines and the lowest average response and turnaround times" when compared to conventional algorithms like Earliest Deadline First (Hassan et al., 2025).

Even under heavy loads, the evolutionary approach maintained optimal performance, demonstrating the power of GAs to find better scheduling strategies than human-designed heuristics.

Similarly, in cloud computing, resource allocation and load balancing have been tackled by evolutionary algorithms. Researchers have used GAs to determine how to distribute workloads among servers or how to migrate tasks in a cloud to reduce load imbalance. These algorithms evaluate configurations based on throughput and resource usage, then evolve the configurations. Studies (e.g., Kumar et al., 2024; Mohammadzadeh & Masdari, 2023) have reported that GA-based scheduling in cloud platforms can reduce execution time and cost by finding more efficient allocations than default strategies (these were summarized in recent reviews). Evolutionary techniques are also often part of hybrid solutions – for instance, a hybrid model might use a GA to globally explore scheduling orders, and a machine learning model to locally predict task execution times or resource needs (Sanjalawe et al., 2025). Such hybrid GA-ML systems leverage the strength of global search (from GA) and the predictive power of ML, leading to very robust optimizers.

2.3 Genetic Programming for Code Optimization:

While GAs usually operate on encoded representations of solutions (like task indices or resource assignments), Genetic Programming (GP) evolves actual programs or code snippets. GP has opened a novel avenue in software optimization known as “genetic improvement of software”. In this approach, the code of a program is modified slightly through evolutionary operations to find a version that runs faster or uses less memory, while still producing correct results. Liou et al. (2024) provide a striking example of this: they developed an automated tool (named GEVO) that uses evolutionary computation to optimize GPU kernel code for scientific applications. The GP system made small code edits to GPU programs (for bioinformatics and simulations) and evaluated their performance on hardware. Over many generations, it evolved versions of the code that achieved 17%–29% reduction in runtime compared to the original, hand-optimized code (Liou et al., 2024). Notably, the discovered code optimizations were non-intuitive – even human GPU experts had not thought of them – highlighting how evolutionary methods can uncover optimization opportunities beyond manual tuning. The success of GEVO and similar genetic programming efforts underscores an exciting point: machine-driven evolutionary search can find performance improvements that humans might miss, due to the enormous space of possible program transformations (Liou et al., 2024). This has implications for



compilers and automated software engineering, where evolutionary algorithms might be integrated to automatically suggest code optimizations for performance.

2.4 Evolutionary vs. Other ML Methods:

One might wonder when to choose evolutionary algorithms over other ML techniques for performance optimization. A rule of thumb emerging from the literature is: if the problem can be framed as a prediction or classification (e.g., predict next memory access, classify a branch as taken/not-taken), then supervised or reinforcement learning is effective; but if the problem is a complex optimization (combinatorial or continuous) with a clear objective but unknown direct mapping, evolutionary algorithms shine. For instance, scheduling and code optimization are well-suited to evolutionary search because they involve exploring permutations or code variants. On the other hand, cache management or prefetching benefit from predictive models (learning what to cache or fetch). There are cases where both are combined, as mentioned earlier, to get the best of both worlds. Evolutionary algorithms can also be more interpretable in some cases – the final solution (schedule or code) is explicit – whereas an opaque neural network might be harder to interpret. However, evolutionary methods can be computationally intensive (evaluating many candidates). Thanks to increasing computing power and parallelism, this is becoming less of a barrier, and the literature shows an uptick in using evolutionary approaches for runtime and memory optimization in recent years (Hassan et al., 2025; Liou et al., 2024; Wang et al., 2023, etc.). Overall, evolutionary algorithms are a powerful tool in the optimizer's toolkit, complementing other ML techniques to push the limits of software performance.

3. ML in Dynamic and Cloud Software Systems

One of the most impactful domains for ML-based optimization is in dynamic, large-scale software systems – notably cloud computing environments and other dynamic systems (such as self-adaptive/autonomic software). These systems are characterized by continuously changing workloads, heterogeneous resources, and the need for real-time decision making. In such contexts, the ability of machine learning to adapt and learn is particularly valuable.

3.1 Dynamic Workloads and Adaptivity:

Dynamic software systems (e.g., online services, cloud applications) experience fluctuations in load and usage patterns. A web service might see surges in traffic, or a



big data pipeline might process varying sizes of data. Static configuration or one-size-fits-all algorithms cannot maintain optimal performance across these changes. Sanjalawe et al. (2025) emphasize that “traditional scheduling algorithms fail to meet the complex requirements of modern cloud infrastructures” and lack adaptability, which motivates AI-driven solutions for their scalability and efficiency (Sanjalawe et al., 2025). Machine learning allows a system to learn from real-time data and past behavior to adjust its strategies on the fly. For example, an AI-based scheduler in a cloud can observe that certain types of tasks always slow down when run together on the same server, and learn to separate them in the future – something a static scheduler wouldn’t do unless explicitly programmed.

3.2 Cloud Resource Management:

Cloud computing provides a fertile ground for ML optimizations because of its scale and variability. Key optimization challenges in clouds include: resource allocation (deciding how many CPU cores, memory, etc., to assign to each service), autoscaling (when to add/remove resources based on demand), and task scheduling across a cluster of machines. Researchers have applied machine learning to all these problems:

- **Resource Demand Prediction:** ML regression models and even deep learning (like LSTM networks) are used to forecast future resource usage (CPU, memory, network) from historical usage data. Accurate prediction enables proactive scaling. For instance, Anbarkhan (2025) used a combination of regression and neural network models to predict cloud workload trends; the system could then dynamically allocate resources ahead of time, avoiding both over-provisioning and under-provisioning (Anbarkhan, 2025). This led to an up to 30% increase in resource utilization and significantly reduced cost, as idle resources were minimized and just enough capacity was provided in advance.
- **Autonomous Autoscaling:** Reinforcement learning, as mentioned, has been deployed for making autoscaling decisions. Unlike fixed threshold rules (e.g., add a server when CPU > 80%), an RL agent learns a nuanced policy that considers multiple metrics and the long-term impact of decisions. AWS and Google Cloud have reported using such AI techniques internally – AWS uses predictive models to scale services and Google’s adaptive RL allocator learns how to respond to shifting workloads (Anbarkhan, 2025). The benefit is more stable performance with lower latency, because the ML system can prevent resources from becoming saturated and also avoid the latency spikes that occur when scaling actions are taken too late.



- **Job Scheduling in Cloud:** Cloud job scheduling decides which tasks run on which machines at what time. AI-driven schedulers can consider a variety of factors (task resource needs, deadlines, data locality, etc.) and optimize for multiple objectives like throughput and energy. A comprehensive review by Sanjalawe et al. (2025) found that AI-based scheduling strategies improved resource allocation, cost efficiency, and energy consumption compared to traditional algorithms, especially in multi-cloud or heterogeneous environments (Sanjalawe et al., 2025). These AI schedulers often use machine learning to predict job runtimes and resource requirements, then use optimization techniques (heuristic or meta-heuristic) to allocate tasks optimally (Sanjalawe et al., 2025). The result is a more balanced system that can handle complex dependencies and varying job sizes without human intervention.

3.3 Case Study – Adaptive Memory Management:

Cloud and dynamic systems are not only about CPU or task scheduling; memory management in such environments is equally important. Consider a managed runtime (like the JVM or a database server) operating under varying loads – it must decide how much memory to give to caching vs. how often to garbage collect, etc. Machine learning can help these systems tune memory usage in real-time. For example, a dynamic memory manager might use an ML model to decide the size of memory pools based on current usage patterns, growing or shrinking caches as needed. Reinforcement learning has been used in buffer cache management where the algorithm learns which pages to keep in memory under changing I/O patterns, leading to improved I/O performance. These approaches ensure that memory is not a static allocation but an elastic resource guided by ML policy.

A specific example from operating systems is using ML for virtual memory page replacement: traditionally OSes use algorithms like LRU to swap pages to disk. But under changing workloads, no single algorithm is optimal. Researchers have created hybrid ML-based page replacement that monitors access patterns and uses a trained model or RL to decide which page to evict. This adaptive method can reduce page-faults and thus improve execution time for memory-intensive applications (Shankar et al., 2025). In general, dynamic systems benefit from ML by gaining the ability to self-optimize in response to environment changes. This fulfills the vision of autonomic computing where systems manage themselves according to high-level goals (here, performance goals) using AI techniques.



3.4 Results in Cloud Context:

To illustrate the impact of ML in a cloud context, we revisit the results of Anbarkhan (2025) and others in a broader perspective. By integrating ML into cloud resource management, systems achieved higher utilization (Figure 1) and also improved application performance metrics such as latency and throughput. An ML-managed cloud can handle higher load without performance degradation, because it smartly distributes work and adds resources just in time. Additionally, from an energy perspective, higher resource utilization means doing the same work with fewer machines, which can reduce power consumption. Many cloud providers are interested in ML for this reason: not just to optimize speed, but to optimize cost and energy, which aligns with green computing goals. Indeed, AI-driven scheduling frameworks have been shown to significantly cut energy usage by finding better task placements (Sanjalawe et al., 2025).

It's also worth noting the complex dependencies in cloud systems – jobs might have dependencies or there might be failures, and AI can help navigate these. For example, if a certain VM shows signs of impending failure (through anomaly detection), an AI scheduler might proactively move tasks away. Such holistic and forward-looking optimization is very difficult to achieve with traditional algorithms that don't learn from data.

In summary, machine learning in dynamic and cloud software systems provides the intelligence required to navigate the complexity and variability inherent in those environments. Whether through predictive scaling, intelligent scheduling, or adaptive memory management, ML techniques ensure that performance (runtime) is optimized and memory/resources are used efficiently under all conditions.

4. Challenges and Future Directions

While the successes of machine learning in optimizing software performance are compelling, there remain several challenges and open research directions. It is important to recognize these to guide future work and to understand the limitations when deploying such solutions in practice.

4.1 Data Availability and Quality:

ML models are only as good as the data they learn from. One challenge is obtaining representative, high-quality data on program behavior or system workload. For new or evolving applications, there may not be enough historical data to train a robust model.



Anbarkhan (2025) points out that data-driven approaches in cloud management require large-scale datasets, and acquiring these (while ensuring they cover all patterns) can be difficult. Moreover, workloads can shift due to external factors (new user behavior, etc.), causing the data used for training to become stale. Future research is looking at techniques like transfer learning and online learning to continually adapt models to new data, and federated learning to leverage data from multiple sources without violating privacy (Anbarkhan, 2025). Ensuring models remain effective as software and usage evolve is an ongoing challenge.

4.2 Performance Overheads and Real-Time Constraints:

Introducing ML logic into a system can introduce computational overhead (CPU cycles to run the model, extra memory to store it). In some contexts, this overhead is negligible, but in others (especially real-time systems), it could be problematic. A key challenge is making sure that the optimization mechanism itself does not slow down the system. Researchers are exploring lightweight models or hardware acceleration (e.g., using a small neural net that can run on a dedicated core or using FPGA/TPU to offload ML tasks) to mitigate this. In real-time systems, any decision-making must meet strict timing constraints. Techniques like bounded-time learning algorithms or using simpler models (like decision trees or linear models) instead of deep networks might be necessary. The good news is that many ML-driven optimizations (like branch prediction, caching) operate on microsecond or nanosecond timescales and have been shown to meet requirements, but careful engineering is needed.

Another aspect is the training time for models. Some solutions might need to retrain models periodically as software changes or new data comes in. If retraining is too slow, the model could lag behind reality. Techniques for faster training or more sample-efficient learning (e.g., few-shot learning for systems) are important future directions.

4.3 Model Accuracy and Robustness:

If an ML model makes a wrong prediction (e.g., mispredicts a branch or prefetches the wrong data), it can hurt performance. While occasional mistakes are tolerable, consistent errors or edge-case failures can negate the benefits of ML. Ensuring robustness of these models – possibly by adding safety checks or combining ML with rule-based logic for critical decisions – is an area of research. There is also interest in explainable AI for systems, where the decisions of a model (like why a resource was allocated) can be understood by engineers, to increase trust in these automated



optimizers (Anbarkhan, 2025). Explainability could help in debugging performance issues when the ML does not behave as expected.

4.4 Integration with Traditional Methods:

Rather than completely replacing existing algorithms, ML-based components often need to coexist with legacy systems or algorithms. Finding the best way to integrate is a challenge: for example, how to smoothly override a database's query optimizer with an ML recommendation, or how to incorporate a learned cache policy into an OS kernel that also has other responsibilities. Compatibility and complexity arise here. One future direction is developing hybrid approaches – e.g., using ML to handle cases where it has high confidence and falling back to traditional methods when uncertain. This hybrid approach can provide a safety net. Sanjalawe et al. (2025) suggest better integration of AI with traditional scheduling as a future focus, to leverage strengths of both (Sanjalawe et al., 2025).

4.5 Scalability and Generalization:

Models that work in a small test environment might face scalability issues in a large production environment. For instance, a scheduling algorithm tested on a 10-node cluster might not scale to a 1000-node cluster without modifications. Future research should ensure that ML solutions scale well (both in terms of computation and the ability to handle more complex environments). Additionally, generalization is key: an ML model trained for one application should ideally adapt or be reusable for similar contexts to reduce re-development effort. Some researchers are investigating meta-learning, where a model learns how to optimize a broad class of programs, not just one – enabling quicker application to new software systems.

4.6 Emerging Technologies:

The landscape of computing is always changing, and new platforms bring new optimization challenges. Edge computing, for example, extends cloud concepts to distributed nodes – scheduling and optimizing across edge devices might require ML to handle network latencies and constrained resources. Blockchain-based systems and serverless computing also pose interesting problems for resource optimization. As noted by Sanjalawe et al. (2025), incorporating emerging technologies (like edge computing and blockchain) into the optimization framework is an open area (Sanjalawe et al., 2025). We might see ML algorithms that optimize not only traditional performance

metrics but also consider security and reliability (for instance, using ML to predict and prevent performance issues caused by security attacks or faults).

4.7 Multi-objective Optimization and Quality of Service:

Finally, future ML optimizers will likely handle multiple objectives simultaneously – not just runtime and memory, but also energy consumption, thermal limits, and QoS guarantees. Multi-objective reinforcement learning or evolutionary algorithms can provide a set of Pareto-optimal solutions (trade-offs), and systems might dynamically shift priorities (for example, favor speed during peak hours, then favor energy saving at night). Some recent works (Wang et al., 2023; Mangalampalli et al., 2023) already deal with multi-objective scheduling using evolutionary algorithms (cited in Sanjalawe et al.'s review), balancing cost, performance, and energy (Sanjalawe et al., 2025). Developing ML models that understand these trade-offs and perhaps even adhere to high-level policies (like a service-level objective) is a promising direction.

In conclusion of this section, the challenges outlined are being actively researched by the community. The consensus is that, despite challenges, AI/ML will play an increasingly central role in software performance engineering. As models become more efficient and data more available, the barriers will diminish. Future research is geared towards making ML-based optimization more scalable, reliable, and easier to integrate, ultimately aiming for self-optimizing systems that require minimal human tuning.

5. Conclusion

Machine learning algorithms have emerged as powerful tools for optimizing runtime performance and memory usage in software systems. Through this review, we have seen that ML techniques – from neural networks and reinforcement learning to evolutionary algorithms – enable systems to learn and adapt in ways that traditional static approaches cannot. They have been successfully applied across a spectrum of scenarios: improving micro-level operations like cache management and branch prediction, optimizing application-level performance through code transformations, and orchestrating macro-level resources in cloud computing. The results reported in the latest research (2021 onward) are impressive: significant reductions in execution time, smarter memory utilization with fewer cache misses, and dynamic resource allocation that boosts efficiency and reduces costs.



Crucially, machine learning brings adaptivity into the system optimization process. As Hyejeong Choi and Sejin Park observed in their 2021 survey, ML-based system optimization is still in its early stages but holds great promise, showing a “wide range of applicability of machine learning technology” for system design (Choi & Park, 2021). The examples and case studies we discussed validate that promise – from 40% faster memory access due to intelligent caching, to genetic algorithms finding scheduling solutions that meet all deadlines, to cloud controllers that cut waste by a quarter through predictive scaling. These improvements translate not only to faster software but often to more economical and energy-efficient operation.

Looking ahead, the integration of ML into software performance tuning is likely to deepen. We can expect future software systems to be built with learning mechanisms as a core component, constantly monitoring and optimizing themselves. This trend aligns with the broader vision of autonomous computing systems that self-optimize, self-heal, and self-configure. There will undoubtedly be hurdles to overcome, such as those discussed in Section 5 – ensuring that these ML systems are robust, secure, and interpretable. But the trajectory of research indicates continuous progress on these fronts.

In closing, the application of machine learning to runtime and memory optimization represents a significant shift in how we approach software performance engineering: from manual, one-time optimizations to automated, continuous improvement processes. By harnessing data and adaptive algorithms, software can achieve levels of performance optimization that were previously unattainable with static methods. This synergy between machine learning and systems optimization is ushering in a new era of intelligent software – software that not only performs well, but learns to perform better over time. The continued collaboration between the machine learning community and systems engineers will be key to realizing the full potential of these approaches. The research reviewed in this paper provides a strong foundation, and we anticipate rapid advancements and real-world deployments in the coming years, ultimately leading to more efficient, faster, and smarter software systems.

6. Acknowledgments

The authors would like to acknowledge the contributions of the academic and research communities whose work laid the foundation for this review. We are particularly grateful to the researchers who have published recent and high-impact studies on the application of machine learning and evolutionary algorithms in software performance optimization. Their findings provided critical insights and data that enriched the depth of this study.



We also thank the anonymous reviewers for their valuable feedback and suggestions, which helped improve the clarity and quality of this manuscript.

7. Appreciation

Special thanks are extended to our affiliated institutions and research centers for providing access to academic databases and computational resources used in the preparation of this review.

We also appreciate the encouragement and guidance of our academic advisors and colleagues, whose constructive discussions and critiques were instrumental throughout the development of this research.

References

Anbarkhan, S. H. (2025). Optimizing Cloud Resource Allocation with Machine Learning: Strategies for Efficient Computing. *Ingénierie des Systèmes d'Information*, 30(1), 1–9.

Choi, H., & Park, S. (2021). A Survey of Machine Learning-Based System Performance Optimization Techniques. *Applied Sciences*, 11(7), 3235.

Hassan, H. E., Ibrahiem, K. H., & Madian, A. H. (2025). Optimizing multiprocessor performance in real-time systems using an innovative genetic algorithm approach. *Scientific Reports*, 15, Article 3842.

Liou, J.-Y., Awan, M., Leyba, K., Šulc, P., et al. (2024). Evolving to Find Optimizations Humans Miss: Using Evolutionary Computation to Improve GPU Code for Bioinformatics Applications. *ACM Transactions on Evolutionary Learning and Optimization*, 4(4), Article 15.

Sanjalawe, Y., Al-E'mari, S., Fraihat, S., & Makhadmeh, S. (2025). AI-driven job scheduling in cloud computing: a comprehensive review. *Artificial Intelligence Review*, 58, Article 197.

Shankar, V., Muralidhar, N., & Krishnamurthy, V. (2025). AI-Assisted Memory and Cache Optimization in Modern Operating Systems and Multicore Processors. *International Journal of Computer Engineering and Technology*, 16(2), 250–264.