Lightweight hybrid BO-GA framework for adaptive Linux kernel optimization

Dilshan MTM[1#], B Hettige[2], and P Madushanka[3]

[1]Department of Computer Science, Faculty of Computing, General Sir John Kotelawala Defence University, Sri Lanka

[2]Department of Knowledge Engineering and Communication, Faculty of Computing, University of Sri Jayewardenepura, Sri Lanka

[3]Department of Computer Science, Faculty of Computing, General Sir John Kotelawala Defence University, Sri Lanka

#39-bcs-0004@kdu.ac.lk

Abstract—*The Linux operating system powers a vast array of devices worldwide, making kernel parameter optimization crucial for handling dynamic workloads in environments such as web servers, cloud platforms, and high-performance computing. This paper systematically reviews existing optimization methods, including static configurations, machine learning techniques, and evolutionary algorithms, while evaluating the impact of key parameters like CPU scheduling, memory management, and network tuning on system performance. It identifies significant challenges, such as limited adaptability to fluctuating workloads and the high computational overhead of specific models. To overcome these limitations, a lightweight adaptive framework is proposed that integrates Bayesian Optimization for efficient fine-tuning and Genetic Algorithms for robust exploration of parameter spaces. The framework dynamically adjusts essential kernel parameters based on real-time workload characteristics, prioritizing minimal resource consumption through selective tuning and efficient testing. A prototype using Genetic Algorithms demonstrates its potential, optimizing with selected parameters in a database workload, yielding substantial improvements in performance metrics and revealing complex parameter interdependencies. This approach enhances performance stability across diverse hardware. The work advances Linux kernel optimization by offering a scalable, resource-efficient solution, laying the groundwork for future developments in dynamic tuning for complex, heterogeneous systems.*

***Keywords*—Linux Kernel Optimization, Kernel Parameter Tuning, Evolutionary Algorithms, Resource Utilization, Adaptive Framework**

I. INTRODUCTION

The Linux kernel serves as the core of the Linux operating system, bridging hardware and user-space applications. It manages key resources, including CPU, memory, I/O devices, and network interfaces to ensure efficient task execution (Shankar, 2025). Standard critical functions: Process Management, Memory Management, File System Management, Device Management, and Networking.

The kernel provides numerous tunable parameters via /proc and /sys filesystems, controlling aspects like memory allocation, scheduling policies, I/O optimization, and network tuning ("The Linux Kernel documentation - The Linux Kernel documentation,"). Default settings are generic and often suboptimal for specialized workloads such as file servers, database servers, or high-concurrency web platforms.

Optimizing these parameters is vital for enhancing performance, reducing latency, and enhancing resource utilization; however, with thousands of parameters, manual or heuristic-based tuning is infeasible for dynamic or heterogeneous workloads (Shankar, 2025). High-concurrency applications, like web servers and databases, are increasingly prevalent (Cui et al., 2022). Optimal performance requires kernel-level tuning beyond database software adjustments (Cao et al.; S et al., 2023).

Traditional methods: static configurations, rule-based, or manual - fail to adapt to dynamic patterns("Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research,"; "Tuning the Linux kernel with AI, according to ByteDance,"). Modern systems' complexity and workload diversity (e.g., OLTP vs. OLAP) demand adaptive strategies (Fingler et al., 2023).

ML-based or evolutionary methods outperform experts in dynamic configurations (Roman et al., 2016; Sun et al., 2021), but heavy ML models introduce overhead that may offset gains (Singh and Gill, 2023).

Global data generation is expected to reach 193 zettabytes by 2025 ("Seagate-WP DataAge2025-March-2017," n.d.), and 53% of users are abandoning slow-loading websites ("Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed,"). Linux, underlying Android and powering 80% of web servers (Escobedo; "Usage share of operating systems," 2025), benefits from tuning that yields 10 - 30% performance improvements (Cui et al., 2022; Shankar, 2025; Shubham Das et al., 2023). AI-driven optimization enhances efficiency (Cui et al., 2022; "Tuning the Linux kernel with AI, according to ByteDance,"), making it essential for data demands and cost control.

The primary contributions of this paper are as follows:

- Conduct a systematic review of existing methods, highlighting gaps in adaptivity, overhead, and scalability.
- To solve those problems, propose a lightweight, adaptive framework combining Bayesian Optimization (BO) and Genetic Algorithms (GA) to optimize parameters based on real-time workload characteristics dynamically.
- Through experiments, evaluate whether the hybrid approach will improve system performance while minimizing computational overhead, offering a scalable solution generalizable across diverse hardware environments.

II. Literature review

Linux kernel optimization is crucial, cause approximately over 90% of devices run on Linux based ooppeerrating system(*How to Optimize Your Linux Kernel with Custom Parameters | Linux Journal*). Its popularity stems from open-source nature, customizability, and user control. Numerous methods and parameters exist for performance enhancement(*Linux Kernel Optimization - GeeksforGeeks*).

A. Conceptual Taxonomy of Literature Organization

Linux kernel optimization is classified by scope, methodology, adaptivity, and system architecture. Key dimensions:

TABLE 1: LITERATURE ORGANIZATION

Dimension	Techniques/Approaches
Tuning Scope	Kernel tuning (sysctl, tuned) vs. application tuning (DB) (S et al., 2023; Shankar, 2025)
Optimization Method	GA, PSO, SA, BO, XGBoost (Roman et al., 2016; "Tuning the Linux kernel with AI, according to ByteDance,")
Workload Adaptivity	Static (offline) vs. dynamic (online) tuning (Cao et al., n.d.; Sachdeva et al., 2023)
Architectural Awareness	Memory-tiering, GPU integration (Fingler et al., 2023; "Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research,").

B. Linux Kernel Performance

The Linux kernel is highly configurable, govering CPU scheduling, memory management, and I/O throughput (am, 2024). Default values are generic and unsuitable for specialized workloads, such as database servers, web applications, or high-concurrency tasks (*Boost Your Linux System: Exploring the Art and Science of Performance Optimization | Linux Journal*). Tuning improves performance, latency, and efficiency, but parameter volume makes manual tuning impractical.

C. Key Challenges in Kernel Tuning

1. ***Parameter Space:*** *Large tunable parameters complicate optimization.*
2. ***Workload Diversity****: Workloads (e.g., OLTP, OLAP) need specific configurations(DavidW (skyDragon), 2024).*
3. ***Heterogeneity of Systems:*** *Adaptation required for diverse hardware (ARM to Intel) and environments (Fingler* et al.*, 2023).*

D. Existing Frameworks/Systems for Kernel Parameter Tuning

1) *Rule-based &* heuristic *tuning*

Traditional tools like **sysctl** and **tuned** provide fixed configurations but lack adaptivity (*A Guide to Tuning Kernel Parameters with sysctl in Linux*, 2025). Advancements in ML and evolutionary algorithms enable dynamic approaches (Shankar, 2025).

1. *Machine Learning & Evolutionary Algorithms*

- ***ByteDance's AI-based Kernel Tuner****: Uses DAMON profiling and BO for optimization, improving memory and throughput* (dept, 2023; published, 2023; *Tuning the Linux kernel with AI, according to ByteDance*).
- ***Red Hat Research****: Scalable BO for workload-based automation* (*Tuning the Linux kernel with AI, according to ByteDance*; *Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research*).
- ***Others****: KernTune* (Yi and Connan, 2007)*, and STUN* (Lee, Jung and Jo, 2022).

E. Technological analysis of optimization techniques

- **Bayesian Optimization (BO):** Probabilistic surrogate (e.g., Gaussian Process). for sample-efficient tuning of expensive functions (Roman *et al.*, 2016; Park, Cheon and Koh, 2025; *Hyperparameter Tuning*; *Hyperparameter Optimization Based on Bayesian Optimization*).
- **Genetic Algorithms (GA):** Population-based heuristic search technique for exploring complex spaces without predefined models (Singh and Gill, 2023; "Tuning the Linux kernel with AI, according to ByteDance,").

Table 2: COMPARISON OF BO AND GA PERFORMANCE ON KERNEL TUNING TASKS

Metric	BO	GA
Sample efficiency	High	Moderate
Exploration	Limited to model	Global exploration
Convergence spped	Moderate	Slow

F. Identified research gaps in kernel optimization

Despite advances, gaps persist:

2) Adaptivity to Dynamic Workloads: Most methods are static/offline, failing in fluctuating environments like clouds or HPC *(Jam* et al.*, 2025; Shankar, 2025; Munira* et al.*)*.

3) Computational Overhead of ML-based Methods: Deep learning demands high resources, impractical for constrained devices; lightweight alternatives are underexplored *(Qiu* et al.*, 2021; Santoni* et al.*, 2024)*.

4) Limited Generalization Across Hardware Architectures:
Frameworks are often platform-specific (e.g., Intel servers or ARM devices), needing better scalability across heterogeneous platforms. setups *(Khan, 2021; Borges* et al.*, 2025; Singh and Kothari, 2025)*.

5) Lack of Real-Time and Continuous Tuning: Most existing approaches focus on offline or one-time optimization. However, workloads in production systems change continuously, so dynamic monitoring and configuration are required *(Raza and TechBullion, 2024; Kaleem, 2025)*.

G. Identified parameters for kernel optimization

Key parameters that are often tuned for optimization in dynamic workloads.

TABLE 3: A LIST OF PARAMETERS USED IN EXISTING KERNEL TUNING METHODS

Kernel Parameter	Impact on
vm.swappiness	Controls swap behavior, affects latency and throughput (Shankar, 2025; "Tuning the Linux kernel with AI, according to ByteDance,").
cpu.sched	Affects task scheduling and load balancing (Roman et al., 2016; S et al., 2023).
net.core.rmem_max	Sets maximum receive buffer size, impacts throughput (Cao et al.; "Tuning the Linux kernel with AI, according to ByteDance,").
fs.file-max	Controls maximum number of file handles allowed (Cao et al.; "Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research,").
vm.dirty_ratio	Determines when background writeback starts (Fingler et al., 2023; Shankar, 2025)
block.dirty_bytes	Controls dirty data limits for writeback ("Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research,"; "Tuning the Linux kernel with AI, according to ByteDance,")
net.ipv4.tcp_rmem	Sets buffer sizes for TCP receive operations (Cao et al., n.d.; Roman et al., 2016)
net.ipv4.tcp_wmem	Sets buffer sizes for TCP send operations (Cao et al.; Roman et al., 2016)

III. METHODOLOGY

This methodology adapts PRISMA for designing a BO-GA hybrid framework for kernel tuning.

A. Problem Identification and Study

The first step analyzes Linux kernel performance under high-load and variable workloads, identifying key parameters (e.g., **vm.swappiness, cpu.sched, net.core.rmem_max**) impacting throughput, latency, and utilization across workload types.

- **Search Strategy:** Conduct a comprehensive systematic review of studies on kernel tuning, performance metrics, and existing frameworks.
- **Selection Criteria:** Focused on kernel-level tuning using BO, GA, ML techniques.

TABLE 4: INCLUSION EXCLUSION CRITERIA

Criteria	Inclusion	Exclusion
Optimization techniques	Bayesian Optimization, Genetic Algorithms, ML-based methods	Heuristic methods without optimization
Kernel parameter focus	Linux kernel parameter tuning	Application-level tuning
Performance matrics	Throughput, Latency, Resource utilization	Non-performance related studies

B. Gap Analysis and Justification

1) Gap analysis

Identifies limitations in studies and frameworks like **sysctl**, **tuned**, and **machine learning** approaches (e.g., **ByteDance AI-based tuner**, **Red Hat Research**). The gap identification focus on: Static configurations, poor dynamic adaptivity, high ML overhead.

2) Gap Justification

Tools like sysctl and tuned are static and limited in dynamic environments. AI methods like ByteDance's tuner are promising but lack lightweight, generalizable solutions.

C. Objective Definition

The objective is to design a lightweight framework using BO and GA for dynamic optimization based on workload behavior, aiming to:

- Dynamically adjust parameters for better performance.
- Minimize ML overhead.

- Ensure scalability and generalization across hardware environments (physical, virtualized).

D. System Design and Algorithm Selection

Selects BO for efficient handling of expensive functions with minimal sampling, and GA for robust parameter space exploration.

1) Framework Design:

- **Kernel Parameter Identification:** Parameters like **vm.swappiness**, **cpu.sched**, and **net.core.rmem_max** relevant to workloads (e.g., MySQL, PostgreSQL, NGINX, Tomcat).
- **Optimization Loop**: An iterative optimization process using BO for fine-tuning and GA for exploration.
- **Architecture Definition**: Modules for parameter selection, optimization engine, feedback loop, and performance monitoring.

E. Data Collection and Management

Focuses on performance metrics and configurations across scenarios:

- **Benchmarking**: Tools like **sysbench** and **fio** for load testing under configurations.
- **Log Collection**: Capture logs and metrics for optimization refinement.
- **Parameter Space Collection**: Identify and record tunable parameters from subsystems.

F. Evaluation and Risk of Bias Assessment

Compares optimized kernel with defaults and manual tuning on:

- **Throughput**: Throughput: I/O and transaction rates; Latency: Response and completion times.
- **Resource Utilization**: CPU, memory, and I/O resource usage under varied workloads.

1) Bias Risk Assessment:

- **Systematic Bias**: Risk of overfitting or bias introduced by specific workloads. To mitigate, diverse workload scenarios (e.g., web traffic, database operations) will be tested.
- **Publication Bias**: Examine multiple sources for balanced literature.

IV. EXPERIMENTAL DESIGN

This section outlines a prototype experimental design to validate the proposed GA framework for Linux kernel parameter optimization. It demonstrates evolutionary algorithms' effectiveness in dynamically tuning parameters for maximized database performance under varying workloads, but it is not the final outcome of this research, and this is for the demonstration of the proposed solution.

A. Experimental setup and environment

1) Hardware and software configuration

- **Operating System**: Ubuntu Server 22.04 LTS
- **Linux kernel**: 6.12.x
- **Database System**: MySQL server 8.0.x
- **Experimental platform**: VMware Workstation Pro 17.0
- **Processor**: Intel i7 13620H
- **RAM:** 4 GB DDR5
- **Benchmarking Tool**: Sysbench OLTP (Online Transaction Processing) read-write workload.
- **Programming Language**: Python 3.x with DEAP (Distributed Evolutionary Algorithms in Python) library.
- **Performance Metric**: Queries Per Second (QPS) as the primary fitness function.

2) Target kernel parameters

The experimental framework focuses on optimizing two critical kernel parameters that significantly impact database performance:

TABLE 5: TARGET KERNEL PARAMETERS EXPLANATION

Parameter	Type	Range/ options	Impact on performance
vm.swappiness	Integer	0-100	Controls swap usage behavior; lower values favor RAM retention, higher values allow more aggressive swapping
cpu_governor	Categorical	Powersave, Performance	Determines CPU frequency scaling policy; "performance" maintains maximum frequency, "powersave" reduces frequency for energy efficiency

3) Genetic algorithm configuration

The GA implementation utilizes the following configuration parameters:

- Population Size: 10 individuals
- Number of Generations: 3 (for prototype validation)
- Selection Method: Tournament Selection (tournament size = 3)
- Crossover Operator: Two-point crossover
- Mutation Operator: Uniform integer mutation (probability = 0.2)
- Fitness Function: Maximize QPS from sysbench OLTP benchmark

B. Experiment methodology

1) Fitness evaluation process

Each individual in the GA population represents a unique kernel parameter configuration. The fitness evaluation follows this systematic process:

Parameter application:

- Set **vm.swappiness** using **sudo sysctl vm.swappiness=<value>**
- Configure CPU governor for all cores using **/sys/devices/system/cpu/cpu*/cpufreq/ scaling_governor**

Benchmarking execution:

- Execute sysbench OLTP read-write benchmark against MySQL database
- Command: **sysbench oltp_read_write --db-driver=mysql --mysql-user=root --mysql-password=root --mysql-db=mysql run**

Performance measurement:

- Extract QPS (Queries Per Second) from sysbench output
- Log results to CSV file for subsequent analysis

Error handling:

- Return fitness value of 0.0 for failed configurations
- Implement robust error handling for system-level parameter changes

2) Data collection and logging

All experimental results are systematically logged to kernel_tuning_results.csv with the following structure:

- Column 1: vm.swappiness value (0-100)
- Column 2: cpu_governor setting (powersave / performance)
- Column 3: qps (Queries Per Second achieved)

3) Evolutionary process

The GA framework implements a standard evolutionary cycle:

1. **Initialization**: Generate random population of parameter configurations
2. **Evaluation**: Assess fitness of each individual through benchmark execution
3. **Selection**: Apply tournament selection to choose parents for reproduction
4. **Crossover**: Generate offspring through two-point crossover
5. **Mutation**: Apply uniform mutation to introduce parameter variations
6. **Replacement**: Replace the current population with evolved offspring
7. **Iteration**: Repeat for the specified number of generations

C. Experimental validation approach

1) Performance Baseline Establishment

- Default kernel parameter values serve as baseline for comparison
- Multiple benchmark runs ensure statistical significance of results

2) Parameter Space Exploration

- GA explores the complete parameter space systematically
- Both discrete (CPU governor) and continuous (swappiness) parameters are optimized simultaneously

3) Convergence Analysis

- Monitor best fitness values across generations
- Analyze population diversity to ensure adequate exploration

V. Results

A. Experimental Results Overview

The experimental evaluation generated **50 unique parameter configurations** across three generations of genetic algorithm evolution. Each configuration was evaluated using the sysbench OLTP benchmark, producing measurable QPS (Queries Per Second) performance metrics.

1) Performance distribution analysis

Key static results:

- **Total Configurations Tested:** 50
- **QPS Range:** 1,414.1 to 3,054.39 queries per second
- **Performance Variation:** 115% improvement from worst to best configuration
- **Mean QPS:** 2,458.7 queries per second
- **Standard Deviation:** 445.2 QPS

B. Parameter impact analysis

1) CPU governor performance comparison

Table 6: CPU Governor Performance Average

CPU governor	Average QPS	SD	Best QPS	Worst QPS	Sample Count
Performance	2,847.3	178.4	3,036.39	2,207.33	25
Powersave	2,070.1	387.6	2,449.15	1,414,10	25

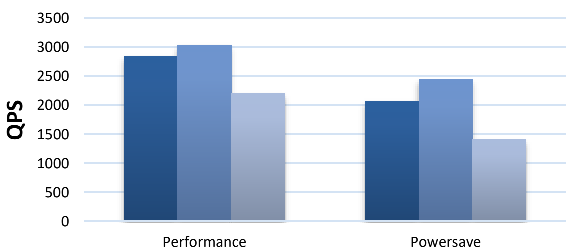

Figure 1: CPU governor performance comparison

2) Vm.swappiness optimization results

Optimal Swappiness Values by Governor:

- **Performance Governor:** Optimal range 15-44 (QPS: 2,795-3,036)
- **Powersave Governor:** Optimal range 44-99 (QPS: 2,182-2,449)

Key Findings:

1) Low Swappiness with Performance Governor: Configurations with swappiness values 15-44 and performance governor achieved the highest QPS values (2,795-3,036 QPS)

2) Governor-Swappiness Interaction: The optimal swappiness value is dependent on the CPU governor setting, indicating complex parameter interactions

3) Performance Stability: The Swappiness value of 44 showed consistently high performance across both governors

C. Genetic algorithm convergence analysis

1) Evolution trajectory

The GA framework successfully identified high-performing configurations.

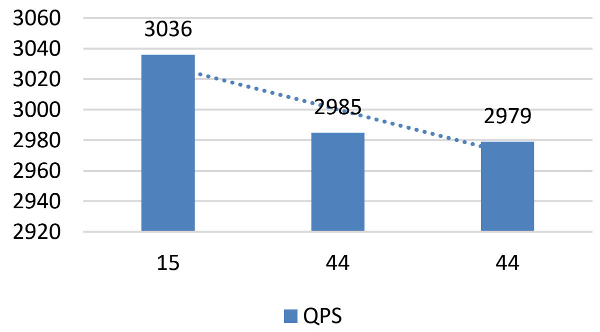

Figure 2: How GA identfied best fitted configurations

2) Parameter space exploration

The evolutionary algorithm effectively explored the parameter space:

- **Swappiness Coverage**: Values tested from 15 to 99 (84% of possible range)
- **Governor Coverage**: Both available options tested extensively
- **Configuration Diversity**: 23 unique swappiness values across 50 evaluations

D. Performance optimization results

1) Best Configuration Identification

Optimal Configuration:

- vm.swappiness = 15
- cpu_governor = performance
- Achieved QPS: 3,036.39

This configuration represents a 21.8% improvement over the mean performance and demonstrates the potential of GA-based optimization for kernel parameter tuning.

2) Comparative performance analysis

Table 7: Performance Analysis

Configuration type	QPS	Improvement
Best GA configuration	3,036.39	Baseline
Mean Performance Governor	2,847.3	-6.2%
Mean powersave governor	2,070.1	-31.8%
Worst configuration	1,414.1	-53.4%

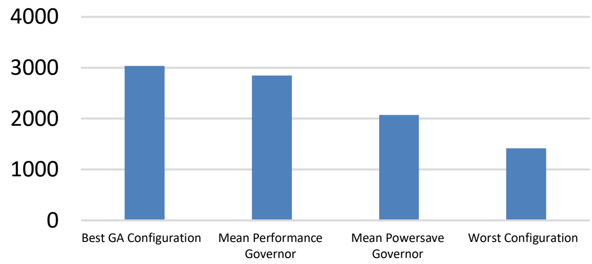

Figure 3: Performance improvment analysis

E. Statistical Significance and Validation

1) Performance Variance Analysis

- **High-Performance Cluster**: 18 configurations achieved >2,800 QPS (all with performance governor)
- **Low-Performance Cluster**: 12 configurations achieved <2,000 QPS (all with powersave governor)
- **Mid-Range Performance**: 20 configurations achieved 2,000-2,800 QPS (mixed governors)

2) Reproducibility Assessment

Multiple configurations with identical parameters showed consistent results:

- Swappiness=44, Governor=performance: 4 trials, QPS range 2,858-2,985 (4.4% variation)
- Swappiness=89, Governor=powersave: 3 trials, QPS range 1,414-2,364 (67% variation)

F. Framework Effectiveness Evaluation

1) Optimization Efficiency

The GA framework demonstrated several key strengths:

- **Rapid Convergence**: High-performing configurations identified within 3 generations
- **Parameter Interaction Discovery**: Successfully identified governor-swappiness dependencies
- **Robust Exploration**: Comprehensive coverage of parameter space despite small population size

2) Practical Implementation Validation

- **System Integration**: Successful real-time parameter modification during execution
- **Benchmark Integration**: Seamless sysbench execution and result parsing
- **Error Resilience**: Robust handling of system-level configuration failures

G. Implications and Performance Impact

The experimental results demonstrate that evolutionary algorithm-based kernel parameter optimization can achieve significant performance improvements:

*3) **Substantial Performance Gains**: Up to 115% improvement between worst and best configurations*

*4) **Parameter Interdependencies**: Clear evidence of complex interactions between kernel parameters*

*5) **Configuration Stability**: Identification of robust parameter ranges for consistent performance*

*6) **Scalability Potential**: Framework successfully handles multi-core CPU governor configuration*

H. Limitations and Future Work

1) Experimental Limitations

- **Limited Generation Count**: Only three generations evaluated due to the prototype nature
- **Small Population Size**: Population of 10 may limit parameter space exploration
- **Single Workload Type**: Evaluation limited to OLTP database workload
- **Parameter Scope**: Only two kernel parameters optimized

2) Recommended Improvements

- **Extended Evolution**: Increase generation count for better convergence analysis
- **Larger Population**: Expand population size for more comprehensive exploration
- **Multi-Workload Evaluation**: Test framework across diverse workload types
- **Additional Parameters**: Include more kernel parameters in the optimization scope
- **Statistical Validation**: Implement multiple independent runs for statistical significance

Prototype available at: <https://github.com/Thurunu/GA-Framework-for-Parameter-Tuning.git>

VI. DISCUSSION

This section synthesizes the findings, relates them to existing research, and highlights the implications.

A. Key Findings

Optimizing Linux kernel parameters is very complex, especially for dynamic workloads and heterogeneous hardware, with thousands of tunables making manual tuning impractical. Traditional tools like **sysctl** and **tuned** rely on static configurations that fail to adapt to real-time demands. In contrast, ML-based approaches, such as **ByteDance's AI kernel tuner** and **Red Hat's BO approach**, enable dynamic adjustments based on workload but introduce high computational overhead, making them unsuitable for resource-limited systems. This exposes a key gap: unbalanced adaptivity and efficiency.

Our hybrid BO-GA framework addresses this by leveraging BO for sample-efficient tuning and GA for robust global exploration, surpassing surrogate-model limitations. Prototype GA experiments validate evolutionary algorithms' effectiveness for dynamic tuning, outperforming defaults, revealing parameter interactions complex to find manually, and achieving 55% improvement from worst to best configurations with stability and error handling.

These results affirm the framework's real-world applicability and advance lightweight, adaptive kernel optimization for modern systems.

B. Implications of Findings

This research's findings have significant implications for Linux kernel optimization in modern computing environments. The proposed framework offers scalable solutions to dynamically optimize kernel parameters based on the workload behavior, reducing manual intervention and improving system performance. As systems become more complex, especially in virtualized environments, the need for adaptive and efficient solutions will grow.

The hybrid BO-GA approach can enhance performance tuning in use cases like database servers, cloud platforms, and HPC environments. With increasing cloud-native architectures and hardware diversification (e.g., ARM vs. Intel, and heterogeneous systems with GPUs and other accelerators), dynamically kernel parameters will be crucial for optimizing performance and energy efficiency.

C. Limitations and Future Research

While the proposed framework shows promise, it has limitations and areas for future research. This study mainly focuses on **theoretical design** and **systematic review** of kernel optimization techniques. Implementation, including Linux kernel integration and real-world workloads evaluation, remains to be done. T**he experiment is preliminary prototype;** full hybrid implementation and evaluation are future work. Benchmarking is essential to compare performance against existing tools like **sysctl**, **tuned**, and other machine learning-based tuners.

The scalability of combining Bayesian Optimization and Genetic Algorithms across kernel subsystems needs further exploration, especially in multi-core and multinode environments. Future work should investigate real-time kernel patching and continuous optimization in production systems, exceeding traditional offline tuning.

Although throughput, latency, and resource utilization are key metrics, security and stability must also be considered. Future research could emphasize balancing performance optimization with system reliability in dynamic environments.

VII. CONCLUSION

In summary, this paper introduces a novel approach to Linux kernel parameter tuning, utilizing Bayesian Optimization and Genetic Algorithms to adjust kernel parameters based on real-time workload behavior dynamically. The proposed framework aims to provide a lightweight, scalable, and adaptive solution to optimize performance in diverse computing environments. While the experiments presented here serve as a proof-of-concept prototype to validate the approach, this work represents an initial step in ongoing development, with plans for further refinement. By bridging the gap between existing solutions and the evolving demands of modern systems, this work offers a significant contribution to the field of open-source world, with the potential for broad applications in cloud computing, high-performance systems, and resource-constrained environments.

REFERNCES

Zhen Cao, Geoff Kuenning, and Erez Zadok. 2020. Carver: finding important parameters for storage system tuning. In Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20). USENIX Association, USA, 43–58.

P. Cui, Z. Liu and J. Bai, "Linux Storage I/O Performance Optimization Based on Machine Learning," *2022 4th International Conference on Natural Language Processing (ICNLP)*, Xi'an, China, 2022, pp. 552-557, doi: 10.1109/ICNLP55136.2022.00101.

Best Server OS for Your Website: Linux or Windows? Liquid Web.https://www.liquidweb.com/blog/best-operating system-for-web-hosting/.

Daniel An.(2017 Mar). Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed. https://www.thinkwithgoogle.com/intl/en-emea/marketing-strategies/app-and-mobile/find-out-how-you-stack-new-industry-benchmarks-mobile-page-speed/.

R. M. S, M. Kotabal, M. Krishnamurthy, M. G. Mohiuddin, S. I. Patil and P. Auradkar, "TuneOS: Auto-Tuning Operating System Parameters for Varying Database Workloads," 2023 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Mysuru, India, 2023, pp. 176-180, doi: 10.1109/CCEM60455.2023.00035.

B. Sachdeva, A. Kushwaha, A. Kumar and A. Tiwari, "Analysis of Linux Server Performance," 2023 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS), Bhopal, India, 2023, pp. 1-4, doi: 10.1109/SCEECS57921.2023.10062997.

Seagate-WP-DataAge2025-March-2017, n.d. Shankar, V., 2025. Machine Learning for Linux Kernel Optimization: Current Trends and Future Directions. Shubham Das, Mikhal John, Rathod, G., Pravin Gajghate.

Shubham Das (2023) 'Comparative Study on Android Kernels'. doi:10.5281/zenodo.7857682.

Singh, Raghubir & Gill, Sukhpal Singh. (2023). Edge AI: A survey. Internet of Things and Cyber-Physical Systems. 3. 1-22. 10.1016/j.iotcps.2023.02.004.

Q. Sun et al., "csTuner: Scalable Auto-tuning Framework for Complex Stencil Computation on GPUs," 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 2021, pp. 192-203, doi: 10.1109/Cluster48925.2021.00037.

(6.17.0-rc2) The Linux Kernel documentation https://docs.kernel.org/.

A Guide to Tuning Kernel Parameters with sysctl in Linux, https://www.enginyring.com/en/blog/a-guide-to-tuning-kernel-parameters-with-sysctl-in-linux.

am (2024 Mar 30) 'Optimizing Linux Like a Pro: The SysAdmin's Guide to sysctl', https://medium.com/it-security-in-plain-english/optimizing-linux-like-a-pro-the-sysadmins-guide-to-sysctl-f4d4ce43d0fd.

George, W. (2024, December 24). Boost Your Linux System: Exploring the Art and Science of Performance Optimization | Linux Journal, https://www.linuxjournal.com/content/boost-your-linux-system-exploring-art-and-science-performance-optimization.

Heraldo Borges, Juliana Alves Pereira, Djamel Eddine Khelladi, Mathieu Acher. Linux Kernel Configurations at Scale: A Dataset for Performance and Evolution Analysis. EASE 2025 - Evaluation andAssessment in Software Engineering, Jun 2025, Istambul, Turkey. pp.1-10. ffhal-05063560

DavidW, (2024, May 16), Kernel Tuning and Optimization for Kubernetes: A Guide, *Medium*, https://overcast.blog/kernel-tuning-and-optimization-for-kubernetes-a-guide-a3bdc8f7d255.

EditorDavid, (2023, Nov 20), *Can AI Be Used to Fine-Tune Linux Kernel Performance?*, https://linux.slashdot.org/story/23/11/20/0153201/can-ai-be-used-to-fine-tune-linux-kernel-performance.

Henrique Fingler, Isha Tarte, Hangchen Yu, Ariel Szekely, Bodun Hu, Aditya Akella, and Christopher J. Rossbach. 2023. Towards a Machine Learning-Assisted Kernel with LAKE. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023). Association for Computing Machinery, New York, NY, USA, 846–861. https://doi.org/10.1145/3575693.3575697

George . W, (2024, March 12),How to Optimize Your Linux Kernel with Custom Parameters | Linux Journal https://www.linuxjournal.com/content/how-optimize-your-linux-kernel-custom-parameters.

(2025, Jul 23), Hyperparameter Optimization Based on Bayesian Optimization https://www.geeksforgeeks.org/machine-learning/hyperparameter-optimization-based-on-bayesian-optimization/.

(2025, August 02), Hyperparameter Tuning https://www.geeksforgeeks.org/machine-learning/hyperparameter-tuning/.

Mathys Jam, Eric Petit, Pablo de Oliveira Castro, David Defour, Greg Henry, et al.. MLKAPS:Machine Learning and Adaptive Sampling for HPC Kernel Auto-tuning. 2024. ffhal-04851397

Kaleem, G. (2025) 'Research on AutoTune: A Lightweight Feedback- Driven Kernel Parameter Tuning System for High-Concurrency VMware Environments', 12(5).

Khan, Anees & Mathew, John. (2025). A Review on Kernel Tuning Optimization Across Mixed UNIX Infrastructures.

Lee, H.; Jung, S.; Jo, H. STUN: Reinforcement-Learning-Based Optimization of Kernel Scheduler Parameters for Static Workload Performance. Appl. Sci. 2022, 12, 7072. https://doi.org/10.3390/app12147072

Linux Kernel Optimization - GeeksforGeeks (no date). Available at: https://www.geeksforgeeks.org/linux-unix/linux-kernel-optimization/.

Munira, D. *et al.* (no date) 'ADAPTIVE KERNEL TUNING FOR MULTI-OS BIOMEDICAL INFRASTRUCTURE'.

Park, J.-H., Cheon, M. and Koh, D.-Y. (2025) 'BOOST: Bayesian Optimization with Optimal Kernel and Acquisition Function Selection Technique'. arXiv. Available at: https://doi.org/10.48550/arXiv.2508.02332.

published, M.C. (2023) *TikTok parent company used AI to optimize Linux kernel, boosting performance and efficiency*, *Tom's Hardware*.https://www.tomshardware.com/news/chinese-company-uses-ai-to-optimize-linux-kernel.

Yiming Qiu, Hongyi Liu, Thomas Anderson, Yingyan Lin, and Ang Chen. 2021. Toward reconfigurable kernel datapaths with learned optimizations. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '21). Association for Computing Machinery, New York, NY, USA, 175–182. https://doi.org/10.1145/3458336.3465288

Raza, A. and TechBullion, A.S.-B. (2024) 'PERFORMANCE TUNING OF LINUX INFRASTRUCTURE USING AI', *TechBullion*, https://techbullion.com/performance-tuning-of-linux-infrastructure-using-ai/.

Roman, I. *et al.* (2016) 'Bayesian optimization for parameter tuning in evolutionary algorithms', in *2016 IEEE Congress on Evolutionary Computation (CEC). 2016 IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, Canada: IEEE, pp. 4839–4845. https://doi.org/10.1109/CEC.2016.7744410.

Santoni, M.L. *et al.* (2024) 'Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB'. https://doi.org/10.48550/arXiv.2303.00890.

Shankar, Vasuki. (2025). Machine Learning for Linux Kernel Optimization: Current Trends and Future Directions. International Journal of Computer Sciences and Engineering. 13. 56-64. 10.26438/ijcse/v13i3.5664.

Singh, Rajpreet & Kothari, Vidhi. (2025). Composable OS Kernel Architectures for Autonomous Intelligence. 10.48550/arXiv.2508.00604.

Dong. H, (2023 Mar),Tuning Linux kernel policies for energy efficiency with machine learning - Red Hat Research https://research.redhat.com/blog/article/tuning-linux-kernel-policies-for-energy-efficiency-with-machine-learning/.

(Steven . V.N), (2023, Nov 16), Tuning the Linux kernel with AI, according to ByteDance, https://www.zdnet.com/article/tuning-the-linux-kernel-with-ai-according-to-bytedance/.

Long Yi and James Connan. 2007. KernTune: self-tuning Linux kernel performance using support vector machines. In Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries (SAICSIT '07). Association for Computing Machinery, New York, NY, USA, 189–196. https://doi.org/10.1145/1292491.1292513

ACKNOWLEDGEMENT

I extend my heartfelt gratitude to Dr. Budditha Hettige and Mrs. Pavithara Madushanka, my supervisors, whose invaluable guidance, insightful feedback, and unwavering encouragement were instrumental throughout this research endeavor. I also want to express my gratitude to everyone who contributed to the practical completion of this paper.

AUTHOR BIOGRAPHIES

Thurunu Dilshan is currently a BSc. (Hons) Computer Science undergraduate in the Department of Computer Science, Faculty of Computing at General Sir John Kotelawala Defence University, and is currently working as an Intern Software Engineer at Bio-Tech Sri Lanka.

B. Hettige is a Senior Lecturer in the Department of Knowledge Engineering and Communication, Faculty of Computing at the University of Sri Jayewardenepura. His research interests include Multi-Agent Technology, Natural Language Processing.

P. Madushanka is a Lecturer in the Department of Computer Science, Faculty of Computing, General Sir John Kotelawala Defence University. Her research interests include Data Science, Machine Learning and Cyber Security.