AutoOS: Make Your OS More Powerful by Exploiting Large Language Models

Huilai Chen [1 2] **Yuanbo Wen** [1] **Limin Cheng** [3] **Shouxu Kuang** [2 3] **Yumeng Liu** [3] **Weijia Li** [2 3] **Ling Li** [3]
Rui Zhang [1] **Xinkai Song** [1] **Wei Li** [1] **Qi Guo** [1] **Yunji Chen** [1]

Abstract

With the rapid development of Artificial Intelligence of Things (AIoT), customizing and optimizing operating system (OS) kernel configurations for various AIoT application scenarios is crucial for maximizing system performance. However, existing approaches falter due to the overwhelming problem complexity (i.e., over 15,000 configuration options in the Linux kernel), together with the huge evaluation costs and error-prone options that may result in OS boot-up failure, which all make it an unresolved problem to optimize the Linux kernel automatically. In this paper, we introduce AutoOS, a novel framework exploiting Large Language Models for customizing and optimizing OS kernel configurations automatically for various AIoT application scenarios. Inspired by the inherently directory-structured kernel configuration process, we first formulate our research problem as optimizing on a dynamic tree. We then propose a novel framework integrating a state machine-based traversal algorithm as the observe-prune-propose-act-correct loop, which can effectively refine the optimization space and ensure a successful OS boot-up. Experimental results show that AutoOS can automatically customize and optimize the OS kernel configurations without human effort. More importantly, AutoOS even achieves better performance by up to 25% than vendor-provided configuration.

[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China [2]University of Chinese Academy of Sciences, Beijing, China [3]Intelligent Software Research Center, Institute of Software, CAS, Beijing, China. Correspondence to: Ling Li <liling@iscas.ac.cn>, Qi Guo <guoqi@ict.ac.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

With the development of AIoT (Artificial Intelligence of Things), the application of embedded systems has become increasingly widespread (Liu et al., 2023b; IoT, 2023). In order to accommodate the growing complexity and diversity of AIoT applications, customizing and optimizing operating system (OS) kernel configurations has become essential for maximizing system performance (Kuo et al., 2020b), particularly in light of embedded devices' constrained hardware resources (Lin et al., 2020). While vendors typically offer default configurations for generality (Kuo et al., 2020a), these presets frequently fail to meet the specific optimization goals, thus presenting substantial opportunities for OS kernel tuning to unlock the full potential of the entire system's capabilities.

However, such performance optimization is so challenging in three-fold. Firstly, the Linux kernel has more than 15,000 configuration options (Oh et al., 2021; Franz et al., 2021), creating a vast and intricate optimization space where options interact with each other and exhibit interdependencies (Mortara & Collet, 2021), making optimizing performance daunting. For example, combining two sets of good performance configuration options directly often results in poor performance. Secondly, the performance evaluation for every configuration is time-consuming (Xia et al., 2023) (e.g., 1$\sim$2 hours), involving several steps like OS kernel compilation, installation, boot-up, and performance testing. Thirdly, as many functional-related options exist, the entire optimization process is error-prone, which may lead to OS boot-up failure. Overall, tuning OS kernel configurations is a complex black-box optimization problem for non-OS professionals who need more expert knowledge.

The difficulty of this problem makes existing approaches including Neural network approaches (Herzog et al., 2021; Martin et al., 2021; Zhang & Huang, 2019) and Bayesian optimization approaches (Jung et al., 2021; Shar et al., 2023) unsurprisingly ineffective. Neural network approaches can learn and adapt to perform a specific task by training a computational model from a well-established dataset (Akgun et al., 2020; Hao et al., 2020; Doudali et al., 2019). However, these approaches require considerable manual ef-

fort to create the dataset from scratch, especially for the time-consuming performance evaluation process for OS kernel configurations. Thus, there is a lack of available datasets to date. Bayesian optimization approaches can usually efficiently find the optimum of an expensive-to-evaluate function by building a probabilistic model that guides the search process (Ziomek & Ammar, 2023; Liu et al., 2023a). However, Bayesian optimization approaches are usually best-suited for optimization over continuous domains of less than 20 dimensions (Frazier, 2018), and thus these approaches still fail to address our problem because of the vast optimization space, which contains more than $2^{15,000}$ possibilities if only considering Boolean options. Furthermore, both Neural network approaches and Bayesian optimization approaches fall short of taking the error-prone options into consideration, which is decisive in guaranteeing the successful boot-up of the OS.

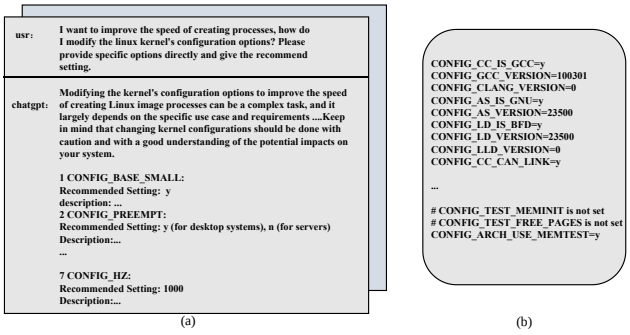

Figure 1. (a) A simple direct request to ChatGPT to modify configuration options for better performance, which is only able to list a smaller number of configuration names, and (b) an example of a vendor-provided default Linux configuration file that only lists the enabled options.

To address the above challenges, we propose to utilize the pre-trained Large Language Models (LLMs) (Touvron et al., 2023; Team et al., 2023; Ouyang et al., 2022; Achiam et al., 2023; Zhang et al., 2022; Jiang et al., 2023; Driess et al., 2023; Li et al., 2023) to optimize the OS performance automatically. However, directly prompting the state-of-the-art LLM (i.e., ChatGPT) to generate relevant OS kernel configuration options for user-specific requirements seems ineffective, as shown in Figure 1(a). This simple attempt shows that it can only make LLMs output a small number (10$\sim$20) of general configuration options, which is not aligned with the user's requirements and means nearly no performance improvement. Besides, it is hard for controlling LLMs to flexibly adapt to specific OS kernel version, since the names and numbers of options in different versions of Linux kernel are slightly different. Additionally, LLMs are not suitable for the natural approach that directly reads the default configuration file of Linux kernels provided by OS distributions or hardware vendors (e.g., Figure 1(b) lists part of the default configurations) and perform further optimization, mainly because of the prompt tokens limits. For example, the state-of-the-art GPT-4 has a 32,000-token context limit (OpenAI, 2023), while the entire Linux kernel configurations contain about 100,0000 words.

To this end, in this work, we propose a novel framework, i.e., AutoOS, unleashing the power of LLMs for customizing and optimizing OS kernel configurations automatically for various AIoT application scenarios. The key insight is that we exploit the expert's prior knowledge in LLMs, which can narrow the vast optimization space aligning with the user requirements, meanwhile detecting and bypassing the error-prone options to guarantee a successful OS boot-up. To achieve this, we first leverage the inherent tree-like structured nature of the OS kernel configurations, formulating this problem as to optimize on a dynamic tree. Then, we propose a state machine-based traversal algorithm with randomness to interact with OS kernel configurations for exploring optimal configuration option combinations. Moreover, we introduce a correcting stage that allows LLMs to automatically debug the OS boot-up failure. Experimental results show that AutoOS achieves 1.08$\times$ to 1.25$\times$ better performance than vendor-provided default configurations. More importantly, AutoOS can automatically customize and optimize the OS kernel configurations without human effort in several hours, significantly reducing the time-consuming costs of manual optimization by experts.

We conclude our contributions as follows:

- To the best of our knowledge, we present AutoOS, the first work to automatically customize and optimize OS kernel configurations by exploiting LLMs.
- Inspired by the inherently directory-structured kernel configuration process, we first formulate the problem of OS kernel configuration as an optimization problem on a dynamic tree. Accordingly, the proposed framework integrates a state machine-based traversal algorithm as the observe-prune-propose-act-correct loop, which can effectively refine the optimization space and guarantee a successful OS boot-up.
- We conduct thorough evaluations on several public Linux distributions, indicating that AutoOS can automatically customize and optimize the OS kernel configurations without human effort. Moreover, AutoOS shows superior performance to those achieved through expert manual optimization, underscoring its efficacy and potential impact in the field.

2. Problem Formulation

In this section, we introduce the formulation of our research problem. We first introduce the background of OS kernel configurations. Then, we propose our problem definition of optimizing on a dynamic tree. Finally, we briefly describe the interaction process between LLMs and OS.

2.1. OS Kernel Configurations Basis

The Linux kernel's 'make menuconfig' command, integral to its build system, enables interactive kernel option configuration (Martin et al., 2021). This command is naturally directory-based, with configuration options often exhibiting complex, sometimes multi-level, dependencies. In the graphical human configuration interface, dependencies dictate that unmet conditions render options unselectable or hidden. Inspired by this, we conceptualize the configuration space as a dynamic tree structure. Changes in option combinations lead to the tree's dynamic transformation, with unselectable options akin to branch removal and newly selectable options to branch insertion. We formalize the research problem as follows:

2.2. Problem Definition: Optimizing on a Dynamic Tree

Definition 2.1. (Optimization on the dynamic tree). Consider the configuration space as a tree $T = (N, E)$, where N represents the nodes (i.e., configuration options), and E represents the relationships between nodes. For a given optimization goal g (i.e., customize and optimize an OS kernel on a specific AIoT device), there exists a special subset of nodes $L_g \subseteq N$, which archives optimal OS performance. Additionally, there is a set $K \subseteq N$, which contains configuration nodes that are related to OS boot-up issues. Given an evaluation function $f : 2^N \to \mathbb{R}$, this function can assign a performance score to any subset of nodes. Formally, the problem is to find the M maximizing $f(M)$, where $M \approx L_g$ and $M \cap K = \emptyset$.

We emphasize the necessity of formulating the problem for optimization on a dynamic tree as follows: Firstly, this tree model is universally applicable across all Linux kernel versions, allowing for an exhaustive exploration of the configuration space as internal nodes expand. Secondly, the dynamic nature of this tree meticulously maintains the inter-option dependencies. Thirdly, the tree's branching structure is inherently conducive to pruning and is also well-suited for conducting searches with a certain degree of randomness, which we will address in Section 3. Lastly, each level's content in the tree can be processed sequentially and the leaf nodes per level are limited, which accommodates the constraints on the context length of LLMs.

2.3. Atomic Action for Interacting between LLMs and OS

It is essential to equip the LLMs with the capability to interact with the OS kernel configurations when optimizing the dynamic tree. Unfortunately, the existing kernel source code does not provide this functionality, only in the form of a graphical interface. We observed that the 'kconfiglib' library (ulfalizer, 2023) provides a command-line form of interaction similar to the interface. To address this issue, AutoOS made slight engineering modifications to the library, abstracting a series of atomic APIs for code-based manipulation of a dynamic tree. These atomic APIs include returning all the semantic information of nodes at a specific level of the tree, modifying a particular configuration, and returning all newly appeared options after configuration changes, *etc*.

3. Method

3.1. Overview

Based on the problem formulation in Definition 2.1, AutoOS introduces a state-machine-based framework to navigate the OS kernel configuration space, i.e., finding the set M in Definition 2.1 to make $f(M)$ while ensuring $M \cap K = \emptyset$, as shown in Figure 2. The framework consists of five stages: observation, pruning, proposing, acting, and correcting. The initial four stages aim to explore potential combinations of kernel configuration options that could enhance performance while avoiding options that impact system boot-up and pruning redundant and irrelevant options. The final correcting stage poses a debug process, which ensures the candidate configurations are viable for a successful OS boot-up. Thus, the above observe-prune-propose-act-correct loop based on LLMs can effectively customize and optimize OS kernel configurations for various AIoT application scenarios.

Concretely, AutoOS employs three key techniques. Firstly, AutoOS introduces the dynamic tree traversal algorithms with randomness, i.e., the former four stages of the state-machine framework, to enhance the search for optimal configuration options within the optimization space. Secondly, to efficiently direct LLMs towards identifying optimized sets of configuration options that meet user requirements, we adopt self-explanation (Rajani et al., 2019) mechanism to fully leverage the LLMs. Lastly, the correction stage enables automatic debugging, swiftly detecting and fixing boot-up failures through the LLMs' expert prior knowledge.

These key techniques will be detailed in the subsequent subsections.

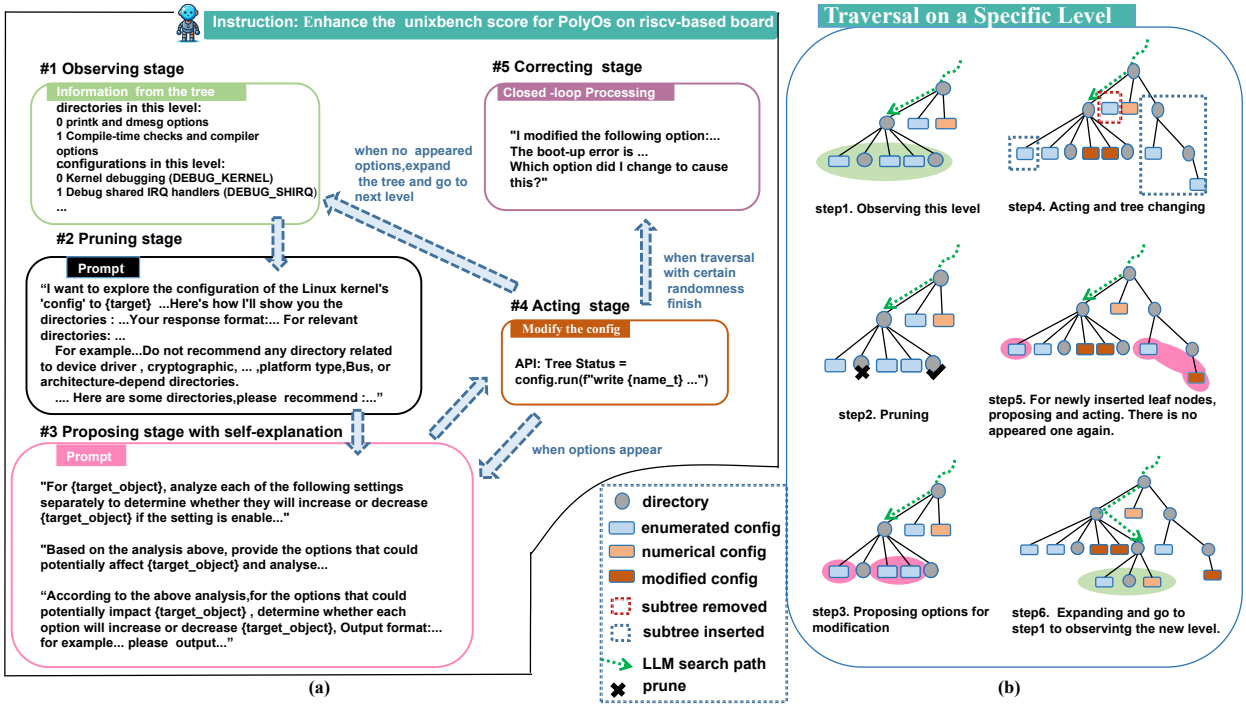

Figure 2. **AutoOS overview.** (a) The proposed AutoOS framework can effectively interact between LLMs and the OS kernel, navigate the OS kernel configuration space, fulfilling the user's requirements for customizing and optimizing an OS kernel for the specific AIoT application scenario. We design an LLM-based systematic prompt template that performs like a state machine, which consists of five stages: **observation**, **pruning**, **proposing**, **acting**, and **correcting**. The initial four stages aim to explore potential combinations of kernel configuration options that could enhance performance while avoiding options that impact system boot-up and pruning redundant and irrelevant options. The final correcting stage poses a debug process, which ensures the candidate configurations are viable for a successful OS boot-up. (b) illustrates an example of how to traverse the dynamic tree on a specific tree level, which will be detailed in Section 3.2.

3.2. Traversal with Certain Randomness

In order to find configuration option combinations M that may enhance performance in the entire optimization space, AutoOS adopts a method of traversing the dynamic tree of configuration options with certain randomness. At the same time, in order to avoid options that affect boot-up and irrelevant options in the combination as much as possible, a pruning strategy is introduced. Algorithm 1 details the pseudocode for this dynamic tree traversal with randomness in AutoOS. Each stage in the traversal algorithm is introduced in detail as follows:

Observing stage. During this stage, atomic operations are adopted to furnish the LLM with comprehensive data regarding all directories D and configuration options S present at this particular level, which including details such as the names and types of the options, as shown in step1 of Figure 2(b). Such a method allows the LLM to adapt to the characteristics of the specific configuration space.

Pruning stage. We direct the LLM to evaluate subdirectories at each tree level, implementing pruning of intermediate nodes. Specifically, for each intermediate node $n_i \in D$, there are two options: pruning (p) and not pruning ($\neg p$). When pruning results,The LLM can determine the probability of pruning as $P(p|n_i)$ based on internal knowledge. It then carries out a sampling process, making the pruning both random and knowledge-guided, as shown in step2 of Fig 2(b). This approach is designed to preemptively exclude exploration into subtrees that lack relevance to performance optimization. Moreover, this pruning process enables AutoOS to sidestep options that could lead to boot-up failures, thereby diminishing the elements within $M \cap K$, significantly reducing the correcting costs in the final correcting stage.

Proposing stage. During this phase, we furnish the LLM with semantic details of options highlighted in pink in Fig 2(b) gathered from the observing and pruning stage, accompanied by comprehensive information (e.g., the range for numerical options) from the kernel source code. For each config option $s_i \in S$, its inclusion in set M and the recommendation of settings for these selected options will be determined based on the probability $P(s_i)$ leveraging

Algorithm 1 Pseudocode for Dynamic Tree Traversal With Certain Randomness in AutoOS

Input: Tree x_i
Initialize $expanded\_set = \{root\}$, $M = \emptyset$
repeat
  Go to the level of $expanded\_set.pop()$
  Get options S, intermediate nodes D for this level
  $expanded\_set.append(LLM\_RandomPrune(D))$
  $propose\_set = LLM\_RandomSelect(S)$
  Push the $propose\_set$ to the execution queue
  $appear\_set = \emptyset$, $disappear\_set = \emptyset$
  repeat
    for $opt\ in\ queue$ **do**
      Modify opt and add to M.
      Update $appear\_set$, $disappear\_set$
      $queue = queue \setminus disappear\_set$
    end for
    $propose\_set = LLM\_RandomSelect(appear\_set)$
    push the $propose\_set$ to queue
  until $propose\_set = \emptyset$
until $expanded\_set = \emptyset$

LLM internal knowledge, which is likely to boost OS performance.

Acting Stage. Following the proposal phase, we enqueue the suggested options for execution, methodically applying the recommendations from the proposing stage. Should alterations occur within the dynamic tree of the configuration space, we exclude the disappeared options from the execution queue. The modified options are marked in red in step4 of Fig 2(b). Subsequently, we revert to the proposing stage to identify any newly emerged options, as shown in step5 of Fig 2(b). This phase, in conjunction with the proposing stage, establishes a cyclical process to manage the continual emergence of options until the dynamic tree stabilizes. Upon completing the traversal of the dynamic tree without any outstanding nodes for expansion, we transition to the final correction stage. If not, we proceed to the next child node, returning to the initial observing stage, as depicted in step6 of Fig 2(b).

By employing randomness in the pruning and proposing phases, AutoOS adeptly traverses various pathways and configurations, culminating in distinctive sets of options M. This strategic incorporation of randomness broadens the exploration scope during the search, thereby augmenting the likelihood of discovering optimal OS configurations.

3.3. Self-Explanation Mechanism

We found that LLMs exhibited inconsistency in articulating the impact of a configuration option on the optimization goal and in determining whether to recommend that option for the same goal. For instance, LLMs might endorse configurations that, according to their interpretation, could negatively affect performance. Such inconsistency leads LLMs to suggest configurations potentially detrimental to the OS kernel's performance.

To address this inconsistency, we introduce a self-explanation mechanism during the proposing stage, illustrated in Figure 2(a). By conducting two rounds of self-explanation, we predominantly achieve a consistent reasoning and decision process for the suggested configurations of LLMs. This mechanism can effectively enhance AutoOS's optimization capabilities. Moreover, by leveraging the LLMs' expert prior knowledge, the self-explanation mechanism avoid modifying redundant options or boot-up failure options, thus significantly streamlining the optimization space and reducing correction efforts.

3.4. Correting Stage

Upon finishing the dynamic tree traversal, we obtain the configuration set M. At this point, the state machine transitions from the acting stage to the correcting stage.

As shown in Fig.2, during this phase, we adopt a closed-loop process, wherein error feedback and constituents of M are relayed to the LLM for boot-up failure detection and correction. The constricted intersection between M and K ($\|M \cap K\|$), a result of the former pruning and self-explanation mechanism, along with the relatively small number of options in M by filtering out the irrelevant options, enables the closed-loop correcting mechanism to efficiently tackle boot-up issues in most scenarios at acceptable costs. When there is only one option causing the startup failure and the closed-loop process is in effect, the correction phase can effectively reduce the debugging overhead from $O(\|M \cap K\|)$ (checking all valid options) or $O(\log \|M \cap K\|)$ (binary search) to $O(1)$.

There are occasions where the LLM-assisted correcting stage fails to detect errors. In such cases, AutoOS uses binary search to identify faulty options, ensuring successful OS boot-up.

4. Evaluation

In this section, we first present the experimental setup. Then, we report our main experiment results, which primarily focus on customizing and optimizing the Linux kernel configuration options for various hardware and scenarios to enhance the overall performance. Besides, we conduct ablation studies to verify the effectiveness of the key techniques (i.e., pruning and self-explanation mechanism) of AutoOS.

4.1. Experimental Setup

We evaluate AutoOS on three different OS kernel configuration tasks. Table 1 lists the details of the OS, option numbers, hardware, scenario, and optimization goal, which

Table 1. **Experimental Setup**. We evaluate AutoOS on three different OS kernel configuration tasks .

	Scenario	OS	Options	Testbed	Optimization Goal
1	AIoT	PolyOS	15,365	Hifive Embedded Board	UnixBench, LMbench
2	AIoT	Fedora	15,561	Hifive Embedded Board	UnixBench
3	General-purpose	Ubuntu	16,621	PC machine	UnixBench

we will detail as follows:

OS. The OS we used include three distinct Linux distributions, i.e., **PolyOS** (PolyOS, 2023), **Fedora**, and **Ubuntu**. Specifically, PolyOS is a lightweight OS designed for the RISC-V architecture. It contains 15,365 OS configuration options with Linux kernel version 5.17.2. Fedora (Fedoraproject, 2023), distinguished for its extensibility and robust community support, is implemented with Linux kernel version 5.18.8, comprising approximately 15,561 options. These distributions primarily support intelligent mobile and AIoT scenarios. Moreover, we also incorporated Ubuntu 22.04.3, a widely-used general-purpose distribution running on Linux kernel version 6.2.0, with over 16,621 configuration options, ensuring a comprehensive evaluation on the general-purpose scenario.

Testbed. The experiment was conducted on two different hardware: an AIoT device and a PC machine. The former is a **Hifive embedded development board** powered by the SiFive Freedom U740 (FU740), an SoC that includes a high-performance multi-core, 64-bit dual-issue, superscalar RISC-V processor with 16GB of DDR4. The latter is a **PC** based on the Intel(R) Core(TM) i7-13700F, x86_64 architecture processor, with 15GB of DDR5 and 30GB of swap memory.

These different kinds of OS distributions and hardware platforms allow us to assess the generalizability and performance of our methods on different OS configuration tasks (e.g., facing the challenge of different Linux kernel versions), providing a comprehensive evaluation of AutoOS's effectiveness.

Benchmark. We adopt the popular **Unixbench scores** (Ge et al., 2020; Xu et al., 2022; Bergman et al., 2023) as the benchmark, as well as the optimization goal of our experiments. UnixBench provides a comprehensive evaluation of an OS's performance across multiple perspectives, such as file I/O, process creation, system function calls, and *etc*..

To further validate the effectiveness of our methods, we also include **LMbench** (McVoy et al., 1996; Kuo et al., 2020b) as a benchmark. This benchmark consists of 54 sub-metric for different OS tasks.

Comparison baselines. We first include the **vendor-provided or default kernel configuration** for each OS as a baseline. Notably, PolyOS's performance on the Hifive Embedded Board, having undergone exhaustive manual optimization, establishes a strong baseline in the expert-level.

The commonly-used approaches including neural networks approaches and bayesian optimization approaches fail to address this problem mainly because of the extensive number of configurations, significant evaluation costs, and the risk of selecting error-prone options leading to OS boot failures. Consequently we consider the **LLM-Vanilla**, i.e., directly prompting LLMs to optimize OS kernel configurations as the second baseline, as shown in Figure 1. Note that LLM-Vanilla with limited prompt length prevents it from reading all the Linux Kernel configuration files, we prompt to directly recommend optimization-related configurations (around 10$\sim$20 options), and handly integrate these to the aforementioned default configurations and evaluate the performance.

Model. In this experiment, we directly integrated the publicly available GPT-3.5-Turbo into AutoOS. To ensure a degree of randomness, the temperature setting for the LLM was set to 1.0.

Seach setting. In order to diversify the configuration options explored during each random traversal of the dynamic tree, AutoOS will automatically explore different optimization targets from Unixbench, including integer or floating-point operations, execl throughput, file transfers, context switches on pipes, process creation throughput, system call capabilities or directly increase the total score of Unixbench. We let AutoOS run the search with 24 search trials with the optimized OS kernel configurations for different OS distributions, and then report the 'best total score' among them. Each search trial is independent and begins at the default configuration. This means that AutoOS is designed to have an upper limit on the total search time (i.e., about one day considering the evaluation costs of about 1 to 2 hours for a candidate OS kernel configuration).

4.2. Overall Performance

Results on AIoT scenarios. Table 2 and Table 3 list the experimental results of the PolyOS and Fedora, respectively, representing the performance improvement of AutoOS on the AIoT scenarios. Results show that AutoOS performs consistently better than the two selected baselines. Concretely, AutoOS archives 1.08$\times$ and 1.26$\times$ performance

Table 2. The best optimization results for PolyOS running on the Sifive Unmatched board, with scores in UnixBench evaluations(higher is better). In the first column, 'default' represents OS configuration from human, 'LLM-Vanilla' corresponds to the naive method from Fig.1's left side. Results show that AutoOS achieves performance improvement than default by 8.4%.

	Dhrystone	Whetstone	Execl	File 1024	File 256	File 4096	Pipe Throughput	Pipe Switching	Process Creation	Shell 1	Shell 8	System Call	Total Score
Default	459	197	515	235	295	213	245	141	166	463	1095	382	309
LLM-Vanilla	461	197	482	221	273	210	213	105	156	436	1023	303	283 (-8.5%)
AutoOS	460	197	**578**	**246**	**364**	**219**	**265**	**175**	**191**	**473**	**1118**	**419**	**335 (+8.4%)**

Table 3. The optimization results for Fedora running on the Sifive Unmatched board, with scores in UnixBench evaluations.

	Dhrystone	Whetstone	Execl	File 1024	File 256	File 4096	Pipe Throughput	Pipe Switching	Process Creation	Shell 1	Shell 8	System Call	Total Score
Default	358	202	197	197	210	167	155	66	98	257	648	369	207
LLM-Vanilla	347	200	176	192	203	182	143	60	89	240	610	293	194 (-6.3%)
AutoOS	**367**	198	**284**	**240**	**310**	**216**	**187**	**137**	**140**	**264**	**696**	**425**	**260 (+25.6%)**

improvement over the vendor-provided default OS kernel configuration, respectively. Notably, although the default OS kernel configuration of PolyOS has undergone exhaustive manual optimization, AutoOS still archives better performance, indicating that AutoOS is able to discover a better OS kernel configuration than experts without human efforts.

LLM-vanilla performs worse than the default OS kernel configuration, contrary to the optimization goals. The reason is two-fold: 1) LLM-Vanilla cannot interact with the entire OS kernel configuration options due to its limited token context windows, and 2) LLM-Vanilla can not handle the interdependencies between different configuration options, which is crucial for performance. AutoOS addresses this through the proposed framework that integrates a state machine-based traversal algorithm on the dynamic tree, which can iteratively recommend options in different tree levels and naturally recognize correlations between options through a tree structure.

Regarding the multiple sub-goals in UnixBench, AutoOS exhibits a higher performance improvement relative to the total score of UnixBnech. For instance, although AutoOS only archives 1.08$\times$ improvement on total score than that of the default kernel on PolyOS, it achieves performance improvement by 24.5% for pipeline switching, 16.7% for process creation, and a notable 23.3% for transfering 256 small data blocks, which is valuable for in real-world applications. In the case of Fedora, alongside a 25.54% performance in the total score, AutoOS accomplishes remarkable enhancements of 44% in execl throughput and 43% in process creation. The modest overall score increment is predominantly due to hardware limitations, particularly as gains in integer and floating-point operations, which are significantly dependent on CPU rather than the OS.

Results on general-purpose scenario. We applied AutoOS to a general-purpose Ubuntu OS running on a PC, further validating the universality of our approach. As shown in Table 4, it can be seen that AutoOS exhibited characteristics similar to those observed on the previous two OS distributions again, reaching a 9.08% increase in overall performance.

In our experiments conducted on PolyOS, Fedora, and Ubuntu, AutoOS executed an average of 0.125, 0.417, and 0.625 correction phases per search across 24 trials. This indicates that the OS generated by AutoOS can successfully boot up in many cases without needing correction phases.

Performance improvement on LMbench. Our previous experiments primarily focused on using Unixbench, a specific benchmark for optimization and evaluation. To verify that our method does not overfit the given optimization goals, we select the OS kernel configuration optimized for the UnixBench on PolyOS, and evaluate its performance for the LMbench. Experimental results show that in the 54 different sub-metrics in LMbench, AutoOS archives better performance for 37 out of 54, ranging from 10% to 30%, and also achieves comparable performance for 13 out of 54 (which mainly consists of integer and floating operations), while only achieves slightly worst performance for 4 out of 54. This experiment provides strong proof that AutoOS does not overfit the optimization goal, and effectively customizes and optimizes the OS kernel configuration to specific hardware.

4.3. Ablation Study

The efficacy of the pruning and self-explanation. We perform an in-depth ablation study to elucidate the efficacy of the pruning and self-explanation methods incorporated in AutoOS. The term 'plain' denotes AutoOS devoid of the pruning and self-explanation features, yet retaining the

Table 4. The optimization results for Ubuntu running on a PC, with scores in UnixBench evaluations.

	Dhrystone	Whetstone	Execl	File 1024	File 256	File 4096	Pipe Throughput	Pipe Switching	Process Creation	Shell 1	Shell 8	System Call	Total Score
Default	6595	2150	1641	6969	4539	13800	3217	952	1448	6015	18554	2351	3885
LLM-Vanilla	6693	2167	1659	6947	4446	13503	3223	985	1483	6033	18868	2286	3898(+0.3%)
AutoOS	**6674**	**2163**	**1800**	**7933**	**5255**	**14536**	**3704**	**989**	**1670**	**6115**	**19609**	**2904**	**4238 (+9.0%)**

Table 5. Ablation studies of pruning (except the first layer in the tree) and the self-explanation proposed in Section 3. The experiment was conducted on PolyOS running on AIoT devices. The 'plain' represents the AutoOS without pruning and self-explanation, but maintains the other component of the proposed framework. Search time encompasses solely the duration of interaction with the LLMs, excluding evaluation periods, whereas correcting time covers the complete debugging and OS evaluation process. The total score is the UnixBench score same as before.

	Plain	*w/* Pruning	*w/* Explanation	AutoOS
Num of modified options	230	20	74	17
Search time (seconds)	321	132	1, 522	429
Num of boot-up options	5	2	0	0
Correcting time (hours)	26.7	6.6	0	0
Total score	137	298	332	335

remainder of the framework's components. To assess the impact of these strategies, we employ several metrics, as delineated in Table 5. These metrics include the total number of suggested options and those causing boot-up failures. Additionally, search time encompasses solely the duration of interaction with the LLMs, excluding evaluation periods, whereas correcting time covers the complete debugging and OS evaluation process. The total score is the UnixBench score same as before.

Experimental results on PolyOS show that both the pruning and self-explanation techniques can effectively reduce the total number of modified options (from 230 to 20 and 74), which refines and narrows the optimization space. Moreover, both two techniques are able to bypass those options that cause boot-up failure. Notably, self-explanation achieves zero boot-up failure options, obviating the need for subsequent correction and markedly reducing AutoOS's total runtime duration, underscoring its effectiveness of significantly streamlining the optimization space and reducing correction efforts. Despite the minimal increased search time necessitated by two rounds of interactions with LLMs, self-explanation secures performance on par with the AutoOS, indicating its effectiveness for optimizing the OS performance.

Different iterations of the self-explanation. We adopt two rounds of self-explanation in AutoOS. The proposed

Table 6. Ablation studies of the number of self-explanation iterations. The 'Iterations' refers to the rounds of self-explanation. The iteration count of 0 indicates that the explanation of an option is not within the same contextual window as the execution result output by the LLM. The 'thinking' denotes the performance impact of an option within the LLM's explanation. The 'doing' represents the actual output when the large model proposes the option. 'Positive', 'Negative' and 'Neutral' denote the three types of impacts that configuration options have on performance.

Iterations	**0**	**1**	**2**	**8**
Positive (doing/thinking)	22/48	25/28	35/36	40/40
Negative	17/29	35/38	37/40	37/39
Neutral	0/59	53/70	55/60	52/57
Consistency (%)	28.60%	83.08%	93.38%	94.85%

Self-Explanation mechanism aims to mitigate inconsistencies in LLMs discussed in Section 3.3. An ablation study was conducted to examine the influence of varying self-explanation iterations on this inconsistency. This study involved a dataset comprising 136 options, all from the pruning phase (i.e., the input to self-explanation) within a single dynamic tree traversal. Configuration options were categorized as positive, negative, or neutral concerning OS performance. Metrics reported include the count of "doing" (proposing a setting for an option), "thinking" (believing in the impact of an option on performance), and overall consistency. We explored several self-explanation iterations: 0, 1, 2, and 8.

The results in Table 6 show that two iterations of self-explanation in AutoOS **1)** effectively enhance the consistency of AutoOS from 28.6% (0 iterations) to 93.38% (2 iterations), and **2)** are enough to address this inconsistency as increasing the iterations of self-explanation from 2 to 8 no longer results in a significant improvement (93.38% $\rightarrow$ 94.85%); instead, it requires a large number of tokens spent in LLMs.

5. Related Work

Data-driven methods. A significant number of studies have concentrated on enhancing software configuration settings through data-driven machine learning approaches. Deepperf (Ha & Zhang, 2019)created a dataset linking application configurations with performance, utilizing FNNs and sparse regularization for performance prediction. VCONF

(Rao et al., 2009)introduced a reinforcement learning-based framework for dynamic VM configuration adjustment in response to online traffic, employing a reward model and an action space. However, these methods are primarily effective in contexts where software performance can be rapidly assessed, which does not apply to scenarios in which OS kernel compilation and installation require 1-2 hours. Additionally, differing from these tasks, the OS kernel comprises several immutable critical configuration options that influence the boot-up process. Herzog(2021) pre-identified 22 kernel parameters from sysfs by which avoid boot-up issues. They employed neural networks to adjust them based on the features of the applications. Notably, the sysfs's kernel parameters can be modified during runtime without necessitating kernel recompilation, thereby reducing the time required for performance evaluation. Acher(2019) used random forests to predict kernel compression sizes, forming a 95,854-entry dataset. Differing from us, they overlooked boot-up issues, focusing on kernel size measurement enabling quick data collection within 10 minutes without the time for installation and evaluation. Our approach also does not aim at kernel size reduction.

Bayesian approach. Bayesian optimization (Bergstra et al., 2011; Snoek et al., 2012) is frequently utilized for costly optimization tasks, particularly excelling in neural network hyperparameter adjustments (Li et al., 2018; Lindauer et al., 2022; Bischl et al., 2023). It is also applied to optimize hardware and software configurations. For example, HiPerBOt (Menon et al., 2020) employs Bayesian optimization to optimize application and platform-level configuration parameters related to high performance. Willemsen (2021) applied it to GPU kernel configuration optimization. In the OS field, Wayfinder(Jung et al., 2021) explored LibOS's network configuration space, choosing over 200 kernel and network library configurations and employing Bayesian optimization. Contrasting with it, our method focus on optimizing across the entire kernel configuration space, without the need for pre-selecting configurations, addressing boot issues and can automatically customize and enhance the overall performance of OS.

Related tools. Genkernel (Thiruvathukal, 2004) is an established tool used to semi-automatically customize hardware-specific kernel configurations on Gentoo and related distributions. But it is only tooling to automate the build process of the Gentoo OS (i.e., automating the kernel's compilation and installation), but rather generates optimized kernel configuration automatically. It provides a very general default configuration and users need to manually adjust the configuration based on their own needs. Conversely, AutoOS automatically optimizes kernel configurations, removing the need for manual customization expertise, and is compatible across various Linux kernel-based OS distributions.

6. Conclusion

In this paper, we introduce AutoOS, a novel framework exploiting Large Language Models for customizing and optimizing OS kernel configurations automatically for various AIoT application scenarios. The proposed framework integrates a state machine-based traversal algorithm as the observe-prune-propose-act-correct loop, which can effectively refine the optimization space and guarantee a successful OS boot-up. Experimental results show that AutoOS can automatically customize and optimize the OS kernel configurations without human effort. Moreover, AutoOS shows superior performance to those achieved through exhaustive manual optimization, underscoring its efficacy and potential impact in the field.

As the first attempt to tackle the challenging problem of customizing and optimizing OS kernel automatically by exploiting LLMs, the future work mainly focuses on two aspects: 1) there are several potential techniques to further improve the performance, e.g., naturally integrating with Bayesian optimization when refining the optimization space by LLMs, and 2) extend the application of AutoOS to other tasks except for the OS kernel performance optimization, such as addressing the challenge of optimizing energy consumption. It is expected that this work will inspire more advanced research in this field.

7. Limitations

AutoOS can be easily adapted to Linux-based OS or those OSs that provide kconfig-like configuration commands. However, it currently can not be directly applied to optimization tasks for non-Linux kernel-based OS. In non-Linux kernel-based systems, the main impediment to AutoOS implementation stems from the lack of readily accessible APIs for abstracting kernel configurations into dynamic trees. Some engineering efforts are required to develop these APIs for automatic optimization and customization of OS kernel configurations.

Acknowledgements

We extend our heartfelt thanks to each individual who contributed to this work during both the submission and rebuttal phases. Additionally, we are grateful to the reviewers for their insightful suggestions that improved the quality of our paper.

This work is partially supported by the National Key R&D Program of China(under Grant 2022YFB4501603), the NSF of China(under Grants 92364202, U22A2028, 61925208, 62222214, 62341411, 62302483, 62102399, 62302482, 62102398, U20A20227, 62372436, 62302478, 62302480), Strategic Priority Research Program of the Chi-

nese Academy of Sciences, (Grant No. XDB0660300, XDB0660301, XDB0660302), CAS Project for Young Scientists in Basic Research(YSBR-029), Youth Innovation Promotion Association CAS, Xplore Prize and Major Program of ISCAS (Grant No. ISCAS-ZD-202402).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning and OS. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Acher, M., Martin, H., Pereira, J. A., Blouin, A., Jézéquel, J.-M., Khelladi, D. E., Lesoil, L., and Barais, O. *Learning very large configuration spaces: What matters for Linux kernel sizes*. PhD thesis, Inria Rennes-Bretagne Atlantique, 2019.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Akgun, I. U., Aydin, A. S., and Zadok, E. Kmlib: Towards machine learning for operating systems. In *Proceedings of the On-Device Intelligence Workshop, co-located with the MLSys Conference*, pp. 1–6, 2020.

Bergman, S., Silberstein, M., Shinagawa, T., Pietzuch, P., and Vilanova, L. Translation pass-through for near-native paging performance in vms. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems (NeurIPS)*, 24, 2011.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.

Doudali, T. D., Blagodurov, S., Vishnu, A., Gurumurthi, S., and Gavrilovska, A. Kleio: A hybrid memory page scheduler with machine intelligence. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pp. 37–48, 2019.

Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Fedoraproject. SiFive unmatched image. https://dl.fedoraproject.org/pub/alt/risc-v/disk_images/Fedora-Developer-Rawhide-20220705.n.0.SiFive.Unmatched/, 2023. Accessed on 2/6/2023.

Franz, P., Berger, T., Fayaz, I., Nadi, S., and Groshev, E. Configfix: interactive configuration conflict resolution for the linux kernel. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 91–100. IEEE, 2021.

Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Ge, X., Niu, B., and Cui, W. Reverse debugging of kernel failures in deployed systems. In *2020 USENIX Annual Technical Conference (USENIX ATC)*, pp. 281–292, 2020.

Ha, H. and Zhang, H. Deepperf: Performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 1095–1106. IEEE, 2019.

Hao, M., Toksoz, L., Li, N., Halim, E. E., Hoffmann, H., and Gunawi, H. S. LinnOS: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 173–190, 2020.

Herzog, B., Hügel, F., Reif, S., Hönig, T., and Schröder-Preikschat, W. Automated selection of energy-efficient operating system configurations. In *Proceedings Of The Twelfth ACM International Conference On Future Energy Systems*, pp. 309–315, 2021.

IoT, A. The rise of industrial ai and aiot: 4 trends driving technology adoption. https://iot-analytics.com/rise-of-industrial-ai-aiot-4-trends-\driving-technology-adoption/, 2023. Accessed on 28/11/2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jung, A., Lefeuvre, H., Rotsos, C., Olivier, P., Oñoro-Rubio, D., Huici, F., and Niepert, M. Wayfinder: towards automatically deriving optimal os configurations. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems (APsys)*, pp. 115–122, 2021.

Kuo, H.-C., Chen, J., Mohan, S., and Xu, T. Set the configuration for the heart of the os: On the practicality of operating system kernel debloating. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–27, 2020a.

Kuo, H.-C., Williams, D., Koller, R., and Mohan, S. A linux in unikernel clothing. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, pp. 1–15, 2020b.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Li, Y., Bubeck, S., Eldan, R., Del Giorno, A., Gunasekar, S., and Lee, Y. T. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., et al. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:11711–11722, 2020.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *The Journal of Machine Learning Research*, 23(1):2475–2483, 2022.

Liu, P., Wang, H., and Qiyu, W. Bayesian optimization with switching cost: Regret analysis and lookahead variants. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4011–4018, 2023a.

Liu, S., Guo, B., Fang, C., Wang, Z., Luo, S., Zhou, Z., and Yu, Z. Enabling resource-efficient aiot system with cross-level optimization: A survey. *IEEE Communications Surveys & Tutorials (COMST)*, 2023b.

Martin, H., Acher, M., Pereira, J. A., Lesoil, L., Jézéquel, J.-M., and Khelladi, D. E. Transfer learning across variants and versions: The case of linux kernel size. *IEEE Transactions on Software Engineering (TSE)*, 48(11):4274–4290, 2021.

McVoy, L. W., Staelin, C., et al. lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pp. 279–294. San Diego, CA, USA, 1996.

Menon, H., Bhatele, A., and Gamblin, T. Auto-tuning parameter choices in hpc applications using bayesian optimization. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 831–840. IEEE, 2020.

Mortara, J. and Collet, P. Capturing the diversity of analyses on the linux kernel variability. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A (SPLC)*, pp. 160–171, 2021.

Oh, J., Yıldıran, N. F., Braha, J., and Gazzillo, P. Finding broken linux configuration specifications by statically analyzing the kconfig language. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 893–905, 2021.

OpenAI. Introducing ChatGPT Enterprise. https://openai.com/blog/introducing-chatgpt-enterprise, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

PolyOS. build_portal. https://gitee.com/riscv-raios/build_portal, 2023. Accessed on 2/6/2023.

Rajani, N. F., McCann, B., Xiong, C., and Socher, R. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

Rao, J., Bu, X., Xu, C.-Z., Wang, L., and Yin, G. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pp. 137–146, 2009.

Shar, L. K., Goknil, A., Husom, E. J., Sen, S., Tun, Y. N., and Kim, K. Autoconf: Automated configuration of unsupervised learning systems using metamorphic testing and bayesian optimization. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1326–1338. IEEE, 2023.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems (NeurIPS)*, 25, 2012.

Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Thiruvathukal, G. K. Gentoo linux: the next generation of linux. *Computing in science & engineering*, 6(5):66–74, 2004.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E.,

Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

ulfalizer. Kconfiglib. https://github.com/ulfalizer/Kconfiglib, 2023. Accessed on 16/11/2023.

Willemsen, F.-J., van Nieuwpoort, R., and van Werkhoven, B. Bayesian optimization for auto-tuning gpu kernels. In *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pp. 106–117. IEEE, 2021.

Xia, Y., Ding, Z., and Shang, W. Comsa: A modeling-driven sampling approach for configuration performance testing. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1352–1363. IEEE, 2023.

Xu, J., Lin, H., Yuan, Z., Shen, W., Zhou, Y., Chang, R., Wu, L., and Ren, K. Regvault: hardware assisted selective data randomization for operating system kernels. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 715–720, 2022.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Y. and Huang, Y. " learned" operating systems. *ACM SIGOPS Operating Systems Review*, 53(1):40–45, 2019.

Ziomek, J. K. and Ammar, H. B. Are random decompositions all we need in high dimensional bayesian optimisation? In *International Conference on Machine Learning (ICML)*, pp. 43347–43368. PMLR, 2023.

A. Possibility of Achieving Positive Results in Experiments

Due to the LLM hallucinations and the complex interplay of OS configuration options under specific hardware and software stacks, AutoOS may not find the optimized configuration in a single search. However, by introducing a degree of randomness and through multiple dynamic tree traversals, it is possible to identify an optimized set of configurations M.

To better illustrate the ability of AutoOS to find good configurations, as well as the impact of randomness, we extend our experimentation on PolyOS. Specifically, we increase the search trials from the initial 24 to 56, and investigate the results of the search process as shown in Figure 3.

Results show that for PolyOs running on the Sifive Unmatched board: 1) AutoOS is able to find the best performance within 24 trials in the experiment. 2) Among the 56 search trials, AutoOS always explores legal configurations, demonstrating the method's robustness. 3) 15 out of 56 (26.78%) surpassed the default performance, indicating that AutoOS can often produce a good result.

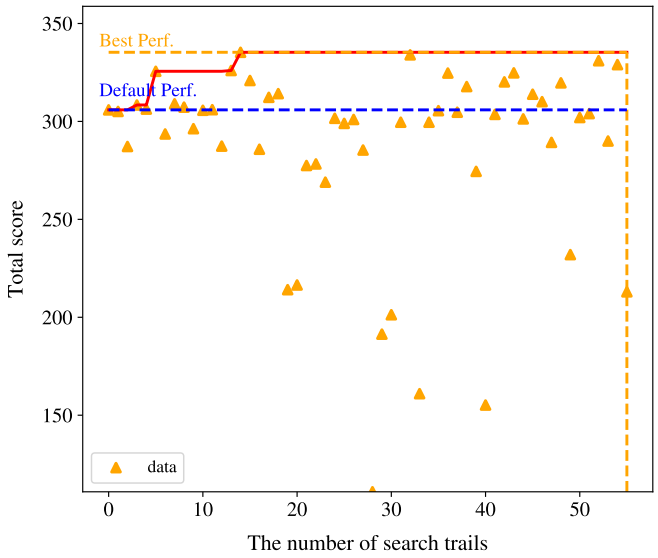

Figure 3. Search exploration to illustrate the possibility of achieving positive results on PolyOS. The horizontal axis represents the iteration number of the search, and 'Total Score' indicates the UnixBench total score of the OS generated during that search. The blue line represents the UnixBench total score of the OS under default configuration, while the red line shows the total score of the optimally generated OS.

B. Case Study of the Optimized OS Kernel Configuration

In this section, we present the configuration option modifications that led to a 25.6% performance enhancement for Fedora in our experiments. In the optimal configuration, AutoOS made modifications to a total of 45 kernel options, starting from the default settings. These adjustments fall into several categories, encompassing continuous memory allocation and management, debugging and security, performance optimization, control groups and resource management, alongside other areas, the impact of which is summarized in Table 7.

It can be seen that, in pursuit of performance optimization, AutoOS disabled analytical features such as `CONFIG_TASKSTATS` and debugging functionalities like `CONFIG_DEBUG_KERNEL`. Additionally, it deactivated options such as `CONFIG_VIRTUALIZATION` and concurrently enabled memory management options like `CONFIG_CONTIGUOUS_MEMORY_ALLOCATOR`.

In the kernel, there are numerous configuration options known for their high overhead, such as KASLR, spectre mitigation, event counters, profiling support, and others. These options often involve trade-offs between functionality, hardware compatibility, and security. To better analyze the behavior of large models regarding the handling of these options, we summarize the modified options related to these features in Table 8.

Table 7. The configuration modifications compared to the default settings that resulted in a 25% performance improvement for Fedora. In the list, a plus sign (+) indicates default disabled settings that are now enabled, while a minus sign (-) indicates the opposite change.

Category	Configuration Options	Modifications in Fedora	Description of options when enabled
Debugging and Security	CONFIG_DEBUG_WQ_FORCE_RR_CPU	+	Force handling of work queues in round-robin fashion for debugging purposes.
	CONFIG_DEBUG_KERNEL	-	Enable kernel debugging features.
	CONFIG_FTRACE	-	Enable function tracing functionality.
	CONFIG_KUNIT	-	Enable kernel unit testing framework.
	CONFIG_RUNTIME_TESTING_MENU	-	Enable runtime testing configuration options
	CONFIG_RCU_CPU_STALL_TIMEOUT	60->3	Set CPU stall timeout for RCU locks.
	CONFIG_BUG_ON_DATA_CORRUPTION	-	Trigger bug reports on data corruption.
	CONFIG_PAGE_POISONING	-	Enable page poisoning to help detect bugs that use released memory.
	CONFIG_DEBUG_RODATA_TEST	-	Test protection of read-only data.
	CONFIG_DEBUG_WX	-	Detect violations of write-execution (WX) permissions in memory regions.
	CONFIG_KFENCE	-	Enable kernel boundary checks to capture errors such as overflows.
	CONFIG_FRAME_POINTER	-	Preserve function frame pointers for stack tracing support.
	CONFIG_FRAME_WARN	2048->0	Set warning threshold for stack frame sizes.
	CONFIG_DYNAMIC_DEBUG	-	Enable dynamic debugging functionality.
	CONFIG_CONSOLE_LOGLEVEL_QUIET	3->4	Set minimum level for console logging.
	CONFIG_MESSAGE_LOGLEVEL_DEFAULT	4->1	Set default message log level.
	CONFIG_SLAB_FREELIST_RANDOM	+	Enable randomization of SLAB free lists.
	CONFIG_LOG_BUF_SHIFT	18->17	Set size of log buffer.
	CONFIG_LOG_CPU_MAX_BUF_SHIFT	12->17	Set maximum CPU log buffer size.
Continuous Memory Allocation and Management	CONFIG_CONTIGUOUS_MEMORY_ALLOCATOR	+	Enable Contiguous Memory Allocator.
	CONFIG_DEFAULT_MMAP_MIN_ADDR	4096->65536	Set minimum address for memory mapping.
	CONFIG_CMA_SYSFS	+	Display CMA information in sysfs.
	CONFIG_DMA_CMA	+	Enable CMA for direct memory access (DMA) allocations.
	CONFIG_DMABUF_HEAPS_CMA	+	Enable CMA's DMA buffer heaps.
	CONFIG_CMA_SIZE_SEL_PERCENTAGE	+	Set CMA size selection based on percentage.
	CONFIG_CMA_ALIGNMENT	null->12	Set alignment requirement for CMA allocations.
	CONFIG_CMA_SIZE_PERCENTAGE	null->0	Set CMA size as a percentage of total memory.
	CONFIG_ZSWAP_COMPRESSOR_DEFAULT_DEFLATE	+	Set default compression algorithm for Zswap to deflate.
	CONFIG_TRANSPARENT_HUGEPAGE_ALWAYS	+	Always use transparent huge pages.
	CONFIG_SLAB_FREELIST_RANDOM	+	Use SLAB memory allocator.
CPU and Power Management	CONFIG_CPU_IDLE	-	Enable CPU idle time management.
	CONFIG_SCHED_AUTOGROUP	-	Enable scheduler automatic grouping functionality.
	CONFIG_PREEMPT_NONE	+	Disable preemptive support.
Profiling	CONFIG_ZSMALLOC_STAT	+	Enable statistics for Zsmalloc.
	CONFIG_PROFILING	-	Enable performance profiling support.
	CONFIG_TASKSTATS	-	Enables the generation of task statistics.
Control Groups and Resource Management	CONFIG_MEMCG	-	Enable memory control groups.
	CONFIG_CGROUP_SCHED	-	Enable control group scheduler.
	CONFIG_CGROUP_PIDS	-	Enable process identifier control
	CONFIG_CGROUP_FREEZER	-	group
	CONFIG_CGROUP_DEVICE	-	Enable control group freezer functionality.
	CONFIG_CGROUP_PERF	-	Enable device control group.
	CONFIG_CGROUP_PERF.	-	Enable performance counter control group.
Others	CONFIG_VIRTUALIZATION	-	Enable support for virtualization technology
	CONFIG_PSI	-	Enables Pressure Stall Information monitoring to track resource bottlenecks

Table 8. Options associated with resource-intensive features like KASLR, spectre mitigation, event counters, and profiling support, among others, for the OS achieving a 25.6% performance improvement on Fedora.

Feature	Option	Modification	Impact on other areas	Impact on Performance
KASLR	SLAB_FREELIST_RANDOM	+	Increases security	Minimal overhead
Spectre Mitigation	no related	N/A	N/A	N/A
Event Counter	BUG_ON_DATA_CORRUPTION	-	May lower immediate detection	Reduce disruptions
	DEBUG_WX	-	May introduce risks	Avoid unnecessary warnings
	ZSMALLOC_STAT	+	Beneficial for monitoring memory	Minimal overhead
	CMA_SYSFS	+	Display CMA information in sysfs	Minimal overhead
Profiling Support	PROFILING	-	Remove profiling functionality	Reduce overhead
	DEBUG_KERNEL	-	Remove debugging functionality	Reducing debugging overhead

Table 8 shows that AutoOS primarily emphasizes optimizing performance when dealing with options related to KASLR, spectre mitigation, event counters, and profiling support. However, it sometimes enables configs with minimal performance overhead in exchange for functionality. For instance, it enables `ZSMALLOC_STAT` and `CMA_SYSFS` for better memory management. Additionally, it activates the `SLAB_FREELIST_RANDOM` option regarding its varied impact on performance, thereby enhancing security.