Can Modern LLMs Tune and Configure LSM-based Key-Value Stores?

Viraj Thakkar, Madhumitha Sukumar, Jiaxin Dai, Kaushiki Singh, and Zhichao Cao
School of Computing and Augmented Intelligence, Arizona State University
{viraj.dt,msukuma2,jdai33,ksingh54,zhichao.cao}@asu.edu

Abstract

Log-Structured-Merge tree-based Key-Value Stores (LSM-KVSs) are important data storage building blocks in modern IT infrastructure. However, tuning their performance involves configuring over 100 parameters, a task typically done manually or with limited parameters in auto-tuning mechanisms. This paper explores and answers the following question: can we leverage LLM's understanding of the system and LSM-KVS components for unrestricted parameter-pool tuning of LSM-KVS?

LLMs are trained on readily available LSM-KVS source code, research papers, and open materials enabling the machines to have human-like understanding. We investigate integrating Large-Language Models (LLMs) into an automated tuning framework for LSM-KVS to enhance the tuning capability and interactivity. Our framework utilizes LLMs to recommend tailored configurations with calibrated prompts based on hardware, system, and workload information. Initial results demonstrate upto **3X** throughput improvements and an upto **9X** reduction in p99 latency across various hardware and workloads compared to the out-of-box configuration for the LSM-KVS.

CCS Concepts: • **Information systems** → **Key-value stores**; **Database utilities and tools**.

Keywords: LSM-KVS, Automatic Tuning and Configuration, Large Language Models

ACM Reference Format:
Viraj Thakkar, Madhumitha Sukumar, Jiaxin Dai, Kaushiki Singh, and Zhichao Cao. 2024. Can Modern LLMs Tune and Configure LSM-based Key-Value Stores?. In *16th ACM Workshop on Hot Topics in Storage and File Systems (HOTSTORAGE '24), July 8–9, 2024, Santa Clara, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3655038.3665954

ACM ISBN 979-8-4007-0630-1/24/07
https://doi.org/10.1145/3655038.3665954

1 Introduction

Log-**S**tructured-**M**erge tree-based **K**ey-**V**alue **S**tores (**LSM-KVS**) have emerged as a fundamental data storage solution in the fast-moving digital world. To serve different use cases, LSM-KVS have been adopted in forms such as RocksDB [9], Big Table [20], LevelDB [10], HBase [2, 16], and Cassandra [1]. An LSM-KVS is engineered with multiple critical components to deliver high performance, including append-only log files, in-memory tables, compaction, flush mechanisms, and Bloom filters. Each component is crucial to improve throughput, storage efficiency, and speed of data retrieval.

The diverse applications of LSM-KVS [8, 11, 18, 33, 39] have been adapted to various setups [15, 24, 31]. To adapt LSM-KVS to diverse applications, popular implementations offer a wide array of configuration parameters, often exceeding 100 as in RocksDB [9] and HBase [2], to manage LSM-KVS' different components. Companies often hire domain experts who understand and interact with the workloads and system configurations that set up the performant LSM-KVS to manage this trade-off for specific workloads [17], storage devices [19, 41, 42, 44], memory devices [27, 28, 45], and deployments [16, 23, 47]. However, with the increasing number of parameters, it is challenging for even the code developers to adequately understand the effect of every option [5, 6, 14].

The growing complexity of LSM-KVS deployments has spurred the development of automated methods for improving performance. These approaches fall into two major categories, **Tuning** - where exposed configuration parameters are tuned to improve performance (e.g. RTune [26], Endure [25], Dremel [48]), and **Optimization** - where the underlying codebase and configuration parameters are both modified to improve performance (e.g. ADOC [46], AC-Key [43]).

In this paper, we focus on Tuning mechanisms. Recent studies like RTune [26], K2VTune [30] and Endure [25] leverage machine learning and optimization to forecast optimal configurations for diverse workloads. Similarly, Dremel [48] and Sami and Eiko [12] propose techniques for online configuration selection and Bayesian optimization, respectively.

However, these approaches only focus on a subset of configuration options (Bloom filters, cache size, etc.) and lack leveraging knowledge of system resources, and workloads. The shortcomings of these approaches highlight the need for a system-aware auto-tuning methodology that demonstrates an understanding of the functionality and correlations in LSM-KVS components. Such a system should be able to tune parameters as needed - not as provided, hence removing the current limitations to auto-tuning only subsets of options.

This research explores Modern Large Language Models (LLMs) for automatic tuning and configuration of LSM-KVS. LLMs have been trained on large datasets comprising websites, blogs, articles, and open-source LSM-KVS code [34, 37, 38]. Given knowledge consumed by modern LLMs, such a tuning approach possesses the necessary components for an un-restricted parameter-pool tuning of LSM-KVS.

LSM-KVS and LLMs have different interfaces, the former based on rigid languages (C++, Java, Python, etc) and the latter utilizing Natural Language. Exploring Such a system presents several challenges: 1) A framework that constructs prompts and facilitates the conversion between code and natural language, and vice versa. 2) Construct parsers to handle output from LLM responses that can be in the form of text, a singular code block, and an interleaving combination of both. 3) Safeguards for unexpected scenarios (e.g.disallow of journaling or logging), and detection of unanticipated responses (e.g. missing options, hallucinated responses).

To explore the LSM-KVS tuning possibilities of using LLM and address the aforementioned challenges, we present **E**lastic **L**arge Language **Mo**del-based Tuning (called **ELMo-Tune**), a novel LLM-based auto-tuning framework for LSM-KVS. ELMo-Tune employs a feedback loop comprising modules dedicated to prompt construction, system resource monitoring, fail-safe management, and interpretation of LLM output for LSM-KVS tuning. The user is only responsible for starting it with an expected system workload (e.g., read intensive, write intensive, and ratios of the same).

We implement a prototype of ELMo-Tune with RocksDB [9] and the GPT-4 API [4], and the framework is open-sourced at **Github** [1] for further investigations and research. Our prototype can achieve up to 3X improvement in throughput and 9X improvement in p99 latency within 7 iterations of tuning compared with the default configurations. We run ELMo-Tune on a large variety of configurations discussed in the evaluation section.

2 Background and Motivation

2.1 LSM-KVS and Its Options

Typical LSM-KVS design involves foreground (Get, Put, Delete, and Scan) and background (flush, Bloom filters, and compaction) components that share system resources. All these components introduce numerous (over 100 sometimes [2, 9]) configuration options to allow adapting an LSM-KVS in different system/workload scenarios. Each configuration option affects system behavior in unique ways, often with dependencies and interactions between different parameters.

[1] https://github.com/asu-idi/ELMo-Tune

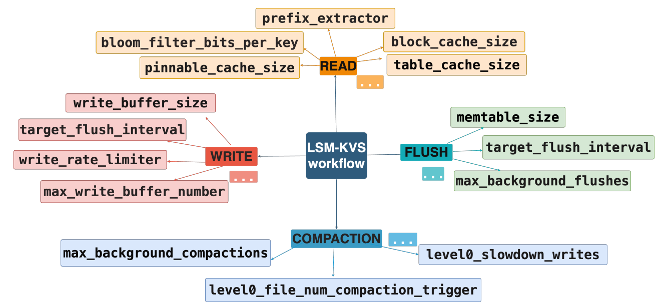


Figure 1. LSM-KVS Stages with Configuration Option

For example, configuring parameters related to compaction, such as *compaction_style* and *max_background_compactions*, counter-intuitively impacts both - compaction and flush, because of how configurations affect resource usage.

Adjusting configuration parameters to suit particular hardware, software, and workload scenarios is complex. Not every parameter equally influences system performance and resource consumption; certain ones, especially those associated with memory management (like write buffer size and active memory usage), data eviction strategies (such as flush triggers), and different compaction approaches (for instance, sub-compaction), may have a more pronounced impact. Figure 1 depicts the vast array of configuration possibilities in LSM-KVS systems.

2.2 Tuning and Configuring LSM-KVS

Manual Tuning Traditionally, LSM-KVS have been tuned manually, relying on the expertise of database administrators. Human experts consider various characteristics of the system (workloads, hardware, and storage devices) to provide tuning configurations [5–7]. However, this approach involves trial-and-error adjustments, consuming valuable time and resources. This approach, while widely used, incurs a high cost every time the system changes and needs meticulous work in cases of fluctuating read-write patterns.

Auto-Tuning Advances

The drawbacks of manual tuning are well known and while no methods replace curated manual tuning, recent advancements in machine learning and optimization algorithms have introduced various auto-tuning techniques for LSM-KVS.

RTune [26] integrates deep learning and genetic algorithms to forecast optimal configurations by analyzing workload patterns. Endure [25] focuses on robust tuning to maximize throughput across diverse scenarios. Dremel [48] adopts a Multi-Armed Bandit model for online configuration selection, while Sami and Eiko [12] incorporate multi-task modeling into a Bayesian optimization framework.

Existing studies have limitations: they often cater to a small subset of options (Bloom filters and Memory buffers

[29, 35], Merging strategies [21, 22, 36]), and lack of understanding of both - system resources and workloads, opting for a trial-and-error approach [25, 26, 48].

2.3 Motivation - LLMs as 'Experts'

Current automated tools focus on a small subset of options, lacking flexibility and intractability with the system and workload. These drawbacks and the lack of transparency often lead to manual tuning approaches which require significant time and resource investments from experts.

To bridge this gap, we explore leveraging Large Language Models (LLMs) in LSM-KVS tuning. LLMs exhibit a combination of human-like and machine-like behaviors, exhibiting a human-like generalized knowledge base and a machine-like lack of downtime. Modern LLMs are trained on extensive datasets that include sources such as websites, tuning guides, research papers, and open-source code repositories like RocksDB, LevelDB, and Cassandra [34, 37, 38]. This training provides LLMs with a comprehensive understanding of LSM-KVS systems and their optimization principles.

The objective of this research is to: 1) explore the depth of understanding that LLMs have of LSM-KVS options; 2) leverage LLM's understanding of system and LSM-KVS components for an un-restricted parameter-pool tuning system for LSM-KVS; 3) evaluate the advantages and limitations of LLM-based tuning framework.

3 Challenges

A Comprehensive Tuning Framework. Designing a system that enables tuning beyond a small subset of options is non-trivial. It necessitates consideration of factors such as software versioning, hardware specifications, and runtime usage. These variables must be effectively integrated and processed by the framework before being relayed to the LLM. Furthermore, the framework should be able to recognize and implement changes suggested by the LLM to the LSM-KVS.

Crafting Effective Tuning Prompts. Given the nuanced nature of language models like ChatGPT, where variations in phrasing can yield very different outputs, formulating a performant and effective prompt is one of the major hurdles. With the abundance of information outputted by LSM-KVS and input limitations of LLMs, questions like, 1) how much information is enough? 2) what information first? and 3) how to formulate the prompt. become increasingly important to answer.

Handling Natural Language Responses. LLMs communicate via Natural Language, while the configuration of LSM-KVS is through much stricter means - often 'ini' files or C++ code. This lack of common language needs special translation and checks that allow the systems to work in the same framework.

Mitigating LLM Hallucinations and Establishing Safeguards. LLMs can occasionally produce confident yet incorrect responses. Therefore, any framework leveraging an LLM must balance trust in its outputs with a healthy degree of skepticism. Robust safeguards are essential to vet LLM-generated suggestions before implementation. Furthermore, certain critical options, such as disabling journaling or I/O flush, should be restricted from modification to prevent performance degradation.

4 ELMo-Tune

In this section, we present **E**lastic **L**arge Language **Mo**del-based Tuning (ELMo-Tune). We design ELMo-Tune to overcome the above challenges and with two major design goals, 1) Allow tuning of all options available for the LSM-KVS, 2) A framework that is flexible to combinations of workloads, hardware, LSM-KVS versions, and storage devices.

4.1 LLMs: Beyond the Subset Paradigm

Large Language Models can be viewed as advanced prediction systems that have been trained on a vast array of publicly accessible internet resources, including blogs, tuning guides, and LSM-KVS source code. Leveraging the knowledge of even the source code of the system can provide a unique advantage. This allows the LLMs to understand the underlying mechanisms and intricacies of the LSM-KVS. Consequently, they can make more informed and precise suggestions for parameter tuning.

Furthermore, LLMs' capability to process and learn from various data sources enables them to configure systems beyond just a subset of parameters. They can analyze the entire configuration space, leading to performance tuning that considers all possible parameter combinations.

4.2 LLM-based Auto-Tuning Framework

In addition to the LLM and LSM-KVS, there are four major modules, 1) Pompt Generator, 2) Option Evaluator, 3) Active Flagger, and 4) Safeguard Enforcer. The orchestration of these components is performed in a feedback loop as shown in Figure 2.

The framework is responsible to orchestrate all modules and ensure they work in unison. The user is responsible for starting the system with an expected system workload (e.g., read intensive, write intensive). ELMo then takes over, implementing the continuous feedback loop between the Benchmarking System and LLM. When a pre-defined stopping criterion is met (e.g., based on minimal performance improvement or a maximum number of iterations), ELMO-Tune outputs the final optimized configuration file.

Prompt Generation ELMo utilizes multiple factors when creating the prompt, combining system information (e.g., via psutil [40] and fio [13]), workload statistics, and current configuration information. The collected information is interlaced together to formulate a prompt.

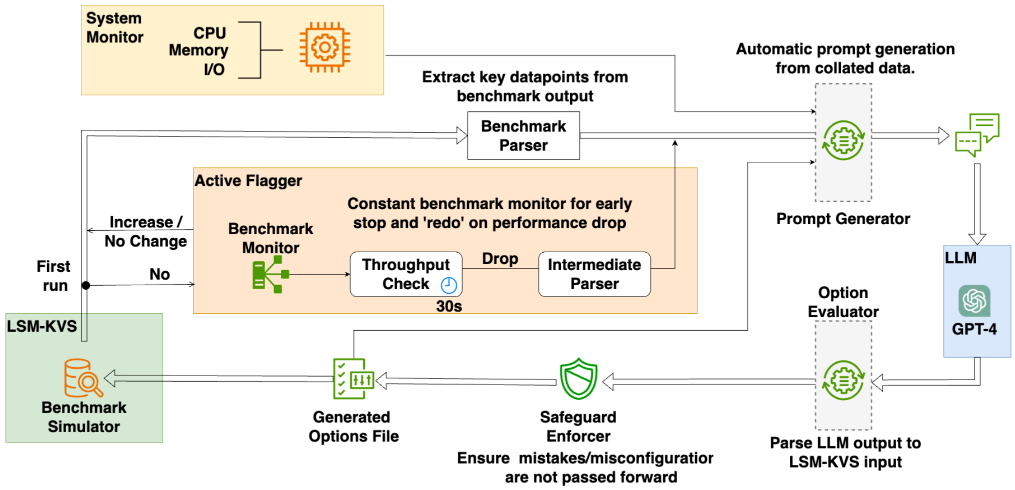


Figure 2. Tuning framework

Option Evaluator The LLM response might take various formats depending on the chosen model and input information. The framework needs to be robust in parsing these responses and extracting the proposed configuration changes.

Active Flagger ELMO-Tune processes the benchmarking results (e.g., extracts the throughput, latency, and tail latency data from the report), compares it with the previous iteration's performance values and determines if the changes enhance performance. If there's an improvement, the new configuration is kept. Otherwise, ELMO-Tune reverts to the previous option file, and makes an intermediate prompt with the information about deterioration, then reruns the benchmark with a new output. This ensures only beneficial changes are recorded, progressively refining the LSM-KVS configuration towards improvement.

Safeguard Enforcer LLMs are susceptible to generating inaccurate or irrelevant outputs, known as hallucinations [32]. ELMo implements two simple and effective solutions that avoid possible mistakes/misconfigurations from LLMs, a configurable blacklist that ensures no necessary options are modified, and a format checker that ensures only specifically formatted LLM output is accepted.

5 Implementation and Evaluation

5.1 Implementation and Experimental Setup

We utilize RocksDB version 8.8.1 [9] and the GPT-4 API [4] for ELMo-Tune prototype development. The codebase of the ELMo-Tune framework is written in Python and publicly available on GitHub [3].

To ensure tests in diverse scenarios, we evaluate ELMo-Tune with the following hardware configurations: CPU (2 CPU Cores, 4 CPU Cores), Memory (4GiB RAM, 8GiB RAM), and Storage Devices (NVMe SSD, and SATA HDD). These different hardware configurations were implemented using different hardware setups in Docker containers. All evaluations were run on the following workloads: 1) Write 50M KV-pairs in random key order (fillrandom (FR) as write-intensive workload); 2) Read 10M KV-pairs in random key order (readrandom (RR) as read intensive workloads) - Database pre-loaded with 25M KV-pairs; 3) 25M ops total of 2 threads doing random-read and random-write (readrandomwriterandom (RRWR) as mix workloads), and 4) 25M ops in Mixgraph, a production workload configured with 50% Writes and 50% Reads [17].

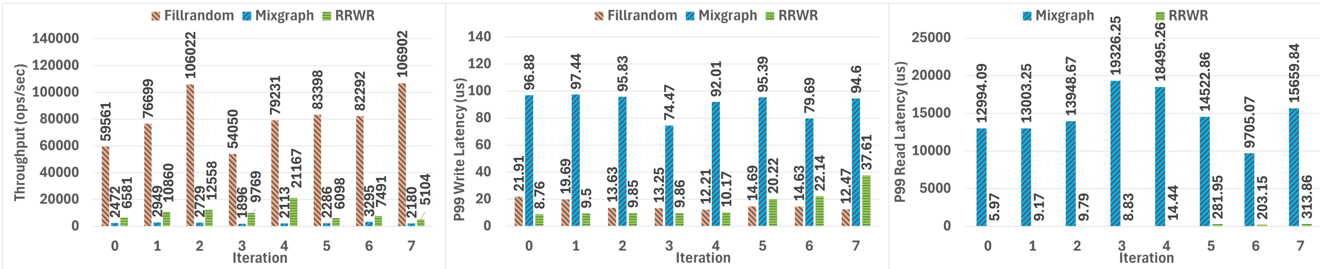
We utilize RocksDB version 8.8.1 in our evaluation and utilize the default configuration provided by db bench, a widely used Rocksdb benchmarking tool, as a baseline for all the tests performed.

5.2 Evaluation Results

We run our evaluation on a variety of configurations, the below subsections go through the results that demonstrate the effectiveness of ELMo.

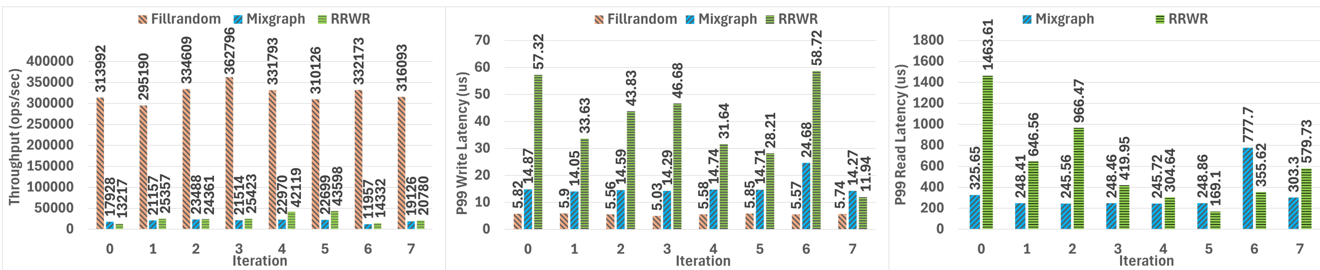
Hardware Configuration. We test ELMo on a total of 4 hardware configurations that vary in CPU (2, 4 cores) and Memory (4, 8 GiB). The results for the Fillrandom test on an NVMe SSD with varying hardware are shown in Table 1 for Throughput and Table 2 for p99 Latency. ELMo works with varied hardware and achieves improvements of 15.5% in throughput with a decrease in p99 latency by 13.5%.

Workloads. When testing ELMo on different workloads, we find it outperforming the default files by as much as 2X in terms of throughput and reducing p99 latency by as much as 9X. The Results for the same have been displayed in Table 3 for Throughput and Table 4 for p99 Latency.



(a) Throughput **(b)** P99 Latency (Write) **(c)** P99 Latency (Read)

Figure 3. Varying workloads on SATA HDD



(a) Throughput **(b)** P99 Latency (Write) **(c)** P99 Latency (Read)

Figure 4. Varying Workloads on NVMe SSD

Table 1. Varying Hardware Configurations for Fillrandom on NVMe SSD - Throughput (ops/sec)

	CPU + Memory (GiB) Config			
	2 + 4	2 + 8	4 + 4	4 + 8
Default	320377	301677	313992	310574
Tuned	362460	348237	362796	329252

Table 2. Varying Hardware Configurations for Fillrandom on NVMe SSD - p99 Latency (us)

	CPU + Memory (GiB) Config			
	2 + 4	2 + 8	4 + 4	4 + 8
Default	5.73	5.92	5.82	5.88
Tuned	5.01	5.42	5.03	5.62

Table 3. Varying Workloads with 4CPUS & 4GiB RAM on NVMe SSD - Throughput (ops/sec)

	Workloads			
	FR	RR	RRWR	Mixgraph
Default	313992	1928	13217	17928
Tuned	362796	5178	43598	23488

Storage Devices. A more comprehensive evaluation is provided for varying storage devices where we show different workloads being tested on two storage devices. This can be seen in Figure 3 for HDDs and Figure 4 for SSDs. Results for Readrandom were discarded as set system limitations have throughputs of <10 ops/sec with tests timing out.

Table 4. Varying Workloads with 4CPUS & 4GiB RAM on NVMe SSD - p99 Latency (us)

	Workloads			
	FR	RR	RRWR	Mixgraph
Default	5.82	2697.55	(Write) 57.32 (Read) 1463.61	(Write) 14.87 (Read) 325.65
Tuned	5.03	1550.2	(Write) 28.21 (Read) 169.10	(Write) 14.59 (Read) 245.56

With Iteration 0 being the default db_bench configuration, we see significant performance improvements over multiple iterations of the test case. This trend is visible both for the SSD and HDD devices. We observe throughput increases of up to 3X and drops in p99 latency of up to 9X.

Changes over Iterations. ELMo posesses the ability to tune and configure all options. Our fillrandom test on SATA HDD for the 2CPUs + 4GiB RAM configuration finds that a total of 23 configuration parameters in RocksDB were tuned by the 7th Iteration. Table 5 shows 15 of these configurations along with how they were changed across iterations.

The results demonstrate how the GPT-4 API iterates and experiments with different configuration parameters to achieve better performance. Furthermore, these parametrs

Table 5. Changes in options over iterations by LLM

Parameter	Default	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	Iteration 7
max_background_flushes	-1	2		1		2	1	2
wal_bytes_per_sync	0	1048576		524288				1048576
bytes_per_sync	0	1048576		524288				1048576
strict_bytes_per_sync	false	true						
max_background_compactions	-1	2		3	2	4	3	
dump_malloc_stats	true	false						
enable_pipelined_write	true	false						
max_bytes_for_level_multiplier	10				8			
max_write_buffer_number	2	3		4	3			6
compaction_readahead_size	2097152		4194304	2097152			4194304	
max_background_jobs	2		4	3		5	4	
target_file_size_base	67108864		33554432	67108864				33554432
write_buffer_size	67108864		33554432	67108864				
level0_file_num_compaction_trigger	4		6	4				
min_write_buffer_number_to_merge	1		2	1			2	3

are modified with consideration of system resources, taking memory and CPU budgets into account. This can be seen in Table 5 when max_background_flushes are set to 2 and also how the total memory budget is maintained in Iteration 1.

6 Conclusion & Discussion

Our present methodology establishes a feedback loop for iterative enhancements and provides notable performance uplifts. Moreover, the methodology can modify and tune a multitude of options, not being limited to a smaller subset.

We observe 1) Adjusting more than 10 options in a sigle iteration leads to marginal improvements, 2) Performing iterations allows the LLM to experiment and learn from past results, 3) Such an approach allows for a flexible tuning approach that can work with a variety of system configurations, workloads, and storage devices, 4) The model responds in patterns similar to online blogs, preferring the same configuration options.

These observations show the potential of such an approach. However, notable limitations are still present - particularly with limited ability to achieve fine-tuning. The LLM model is particularly good at providing a jumpstart to configuration. A solution that leverages this property, in cohesion with fine-tuning mechanisms would enable faster and potentially better tuning. Furthermore, the LLM has limitations in terms of recognizing newer and depracated options, this needs special attention as modern options (e.g. Dynamic level sizing in Rocksdb) that are more useful can often be overlooked, and older options (e.g Flush Job Count) can be unncessarily focused upon.

A methodology like ELMo promises configurational flexibility and unbounded tuning capability, and while the limitations exist, we believe that they can be overcome and the approach shows promise for further research.

Acknowledgements

We would like to thank our anonymous reviewers for their valuable feedback. We thank all the members of ASU-IDI Lab for providing useful comments. This work was partially funded by the Arizona State University startup fund.

References

[1] [n.d.]. Apache Cassandra | Apache Cassandra Documentation. https://cassandra.apache.org/_/index.html
[2] [n.d.]. Apache HBase – Apache HBase™ Home. https://hbase.apache.org/
[3] [n.d.]. asu-idi/ELMo-Tune. https://github.com/asu-idi/ELMo-Tune
[4] [n.d.]. GPT-4 API general availability and deprecation of older models in the Completions API. https://openai.com/blog/gpt-4-api-general-availability
[5] [n.d.]. Navigating the Minefield of RocksDB Configuration Options | by Kartik Khare | Better Programming. https://betterprogramming.pub/navigating-the-minefield-of-rocksdb-configuration-options-246af1e1d3f9
[6] [n.d.]. RocksDB Tuning Guide. https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide
[7] [n.d.]. RocksDB* Tuning Guide on Intel® Xeon® Processor Platforms. https://www.intel.com/content/www/us/en/developer/articles/guide/rocksdb-tuning-guide-on-xeon-based-system.html
[8] 2021. RocksDB in Microsoft Bing. https://blogs.bing.com/Engineering-Blog/october-2021/RocksDB-in-Microsoft-Bing
[9] 2024. facebook/rocksdb. Meta. https://github.com/facebook/rocksdb original-date: 2012-11-30T06:16:18Z.
[10] 2024. google/leveldb. https://github.com/google/leveldb original-date: 2014-08-27T21:17:52Z.
[11] 2024. Tencent/paxosstore. https://github.com/Tencent/paxosstore original-date: 2017-08-25T09:00:19Z.
[12] Sami Alabed and Eiko Yoneki. 2021. High-Dimensional Bayesian Optimization with Multi-Task Learning for RocksDB. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (Online, United Kingdom) *(EuroMLSys '21)*. Association for Computing Machinery, New York, NY, USA, 111–119. https://doi.org/10.1145/3437984.3458841
[13] Jens Axboe. 2022. Flexible I/O Tester. https://github.com/axboe/fio original-date: 2012-10-22T08:20:41Z.
[14] Mikhail Bautin, Kannan Muthukkaruppan, and Mikhail Bautin and Kannan Muthukkaruppan. 2019. Enhancing RocksDB for Speed and Scale | YugabyteDB. https://www.yugabyte.com/blog/enhancing-

rocksdb-for-speed-scale/

[15] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. 2020. Hailstorm: Disaggregated Compute and Storage for Distributed LSM-based Databases. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 301–316. https://doi.org/10.1145/3373376.3378504

[16] Zhichao Cao, Huibing Dong, Yixun Wei, Shiyong Liu, and David HC Du. 2022. IS-HBase: An In-Storage Computing Optimized HBase with I/O Offloading and Self-Adaptive Caching in Compute-Storage Disaggregated Infrastructure. *ACM Transactions on Storage (TOS)* 18, 2 (2022), 1–42.

[17] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. 2020. Characterizing, Modeling, and Benchmarking {RocksDB} {Key-Value} Workloads at Facebook. 209–223. https://www.usenix.org/conference/fast20/presentation/cao-zhichao

[18] Zhang Cao, Chang Guo, Ziyuan Lv, Anand Ananthabhotla, and Zhichao Cao. 2024. SAS-Cache: A Semantic-Aware Secondary Cache for LSM-based Key-Value Stores. In *38th Intl. Conf. on Massive Storage Systems and Technology*.

[19] Zhichao Cao, Hao Wen, Fenggang Wu, and David HC Du. 2023. SMRTS: A Performance and Cost-Effectiveness Optimized SSD-SMR Tiered File System with Data Deduplication. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 275–282.

[20] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (jun 2008), 26 pages. https://doi.org/10.1145/1365815.1365816

[21] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 79–94. https://doi.org/10.1145/3035918.3064054

[22] Niv Dayan and Stratos Idreos. 2018. Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging. In *ACM SIGMOD International Conference on Management of Data*.

[23] Siying Dong, Shiva Shankar P, Satadru Pan, Anand Ananthabhotla, Dhanabal Ekambaram, Abhinav Sharma, Shobhit Dayal, Nishant Vinaybhai Parikh, Yanqin Jin, Albert Kim, et al. 2023. Disaggregating RocksDB: A Production Experience. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–24.

[24] Haoyu Huang and Shahram Ghandeharizadeh. 2021. Nova-LSM: A Distributed, Component-based LSM-tree Key-value Store. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 749–763. https://doi.org/10.1145/3448016.3457297

[25] Andy Huynh, Harshal A. Chaudhari, Evimaria Terzi, and Manos Athanassoulis. 2022. Endure: a robust tuning paradigm for LSM trees under workload uncertainty. *Proc. VLDB Endow.* 15, 8 (apr 2022), 1605–1618. https://doi.org/10.14778/3529337.3529345

[26] Huijun Jin, Jieun Lee, and Sanghyun Park. 2022. RTune: a RocksDB tuning system with deep genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 1209–1217. https://doi.org/10.1145/3512290.3528726

[27] Hiwot Tadese Kassa, Jason Akers, Mrinmoy Ghosh, Zhichao Cao, Vaibhav Gogte, and Ronald Dreslinski. 2022. Power-optimized Deployment of Key-value Stores Using Storage Class Memory. *ACM Transactions on Storage (TOS)* 18, 2 (2022), 1–26.

[28] Hiwot Tadese Kassa, Jason Akers, Mrinmoy Ghosh, Zhichao Cao, Vaibhav Gogte, and Ronald G Dreslinski. 2021. Improving Performance of Flash Based Key-Value Stores Using Storage Class Memory as a Volatile Memory Extension.. In *USENIX Annual Technical Conference*. 821–837.

[29] Taewoo Kim, Alexander Behm, Michael Blow, Vinayak Borkar, Yingyi Bu, Michael J. Carey, Murtadha Hubail, Shiva Jahangiri, Jianfeng Jia, Chen Li, Chen Luo, Ian Maxon, and Pouria Pirzadeh. 2020. Robust and efficient memory management in Apache AsterixDB. *Software: Practice and Experience* 50, 7 (2020), 1114–1151. https://doi.org/10.1002/spe.2799 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2799.

[30] Jieun Lee, Sangmin Seo, Jonghwan Choi, and Sanghyun Park. 2024. K2vTune: A workload-aware configuration tuning for RocksDB. *Information Processing & Management* 61, 1 (2024), 103567.

[31] Jianchuan Li, Peiquan Jin, Yuanjin Lin, Ming Zhao, Yi Wang, and Kuankuan Guo. 2021. Elastic and Stable Compaction for LSM-tree: A FaaS-Based Approach on TerarkDB. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 3906–3915. https://doi.org/10.1145/3459637.3481913

[32] Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. 2024. Exploring and Evaluating Hallucinations in LLM-Powered Code Generation. https://doi.org/10.48550/arXiv.2404.00971 arXiv:2404.00971 [cs].

[33] Gaoji Liu, Chongzhuo Yang, Qiaolin Yu, Chang Guo, Wen Xia, and Zhichao Cao. 2024. Prophet: Optimizing LSM-Based Key-Value Store on ZNS SSDs with File Lifetime Prediction and Compaction Compensation. In *38th Intl. Conf. on Massive Storage Systems and Technology*.

[34] Yiheng Liu, Hao He, Tianle Han, Xu Zhang, Mengyuan Liu, Jiaming Tian, Yutong Zhang, Jiaqi Wang, Xiaohui Gao, Tianyang Zhong, Yi Pan, Shaochen Xu, Zihao Wu, Zhengliang Liu, Xin Zhang, Shu Zhang, Xintao Hu, Tuo Zhang, Ning Qiang, Tianming Liu, and Bao Ge. 2024. Understanding LLMs: A Comprehensive Overview from Training to Inference. https://doi.org/10.48550/arXiv.2401.02038 arXiv:2401.02038 [cs].

[35] Chen Luo and Michael J. Carey. 2020. Breaking down memory walls: adaptive memory management in LSM-based storage systems. *Proceedings of the VLDB Endowment* 14, 3 (Nov. 2020), 241–254. https://doi.org/10.14778/3430915.3430916

[36] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 631–645. https://doi.org/10.1145/3183713.3196908

[37] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large Language Models: A Survey. https://doi.org/10.48550/arXiv.2402.06196 arXiv:2402.06196 [cs].

[38] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni,

Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, C. J. Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs].

[39] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Shiva Shankar P, Mike Shuey, Richard Wareing, Monika Gangapuram, Guanglei Cao, Christian Preseau, Pratap Singh, Kestutis Patiejunas, JR Tipton, Ethan Katz-Bassett, and Wyatt Lloyd. 2021. Facebook's Tectonic Filesystem: Efficiency from Exascale. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. USENIX Association, 217–231. https://www.usenix.org/conference/fast21/presentation/pan

[40] Giampaolo Rodola. 2024. giampaolo/psutil. https://github.com/giampaolo/psutil original-date: 2014-05-23T14:01:48Z.

[41] Fenggang Wu, Bingzhe Li, Zhichao Cao, Baoquan Zhang, Ming-Hong Yang, Hao Wen, and David HC Du. 2019. ZoneAlloy: Elastic Data and Space Management for Hybrid SMR Drives. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*.

[42] Fenggang Wu, Bingzhe Li, Baoquan Zhang, Zhichao Cao, Jim Diehl, Hao Wen, and David HC Du. 2020. Tracklace: Data management for interlaced magnetic recording. *IEEE Trans. Comput.* 70, 3 (2020), 347–358.

[43] Fenggang Wu, Ming-Hong Yang, Baoquan Zhang, and David H. C. Du. 2020. {AC-Key}: Adaptive Caching for {LSM-based} {Key-Value} Stores. 603–615. https://www.usenix.org/conference/atc20/presentation/wu-fenggang

[44] Fenggang Wu, Baoquan Zhang, Zhichao Cao, Hao Wen, Bingzhe Li, Jim Diehl, Guohua Wang, and David HC Du. 2018. Data management design for interlaced magnetic recording. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*.

[45] Ting Yao, Yiwen Zhang, Jiguang Wan, Qiu Cui, Liu Tang, Hong Jiang, Changsheng Xie, and Xubin He. 2020. {MatrixKV}: Reducing Write Stalls and Write Amplification in {LSM-tree} Based {KV} Stores with Matrix Container in {NVM}. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 17–31.

[46] Jinghuan Yu, Sam H. Noh, Young-ri Choi, and Chun Jason Xue. 2023. {ADOC}: Automatically Harmonizing Dataflow Between Components in {Log-Structured} {Key-Value} Stores for Improved Performance. 65–80. https://www.usenix.org/conference/fast23/presentation/yu

[47] Qiaolin Yu, Chang Guo, Jay Zhuang, Viraj Thakkar, Jianguo Wang, and Zhichao Cao. 2024. CaaS-LSM: Compaction-as-a-Service for LSM-based Key-Value Stores in Storage Disaggregated Infrastructure. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[48] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2022. Dremel: Adaptive Configuration Tuning of RocksDB KV-Store. In *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (Mumbai, India) *(SIGMETRICS/PERFORMANCE '22)*. Association for Computing Machinery, New York, NY, USA, 61–62. https://doi.org/10.1145/3489048.3530970