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

**Viraj Thakkar**, Madhumitha Sukumar, Jiaxin Dai, Kaushiki Singh, Zhichao Cao

***INTELLIGENT DATA INFRASTRUCTURE LAB (ASU-IDI)***

***Arizona State University***

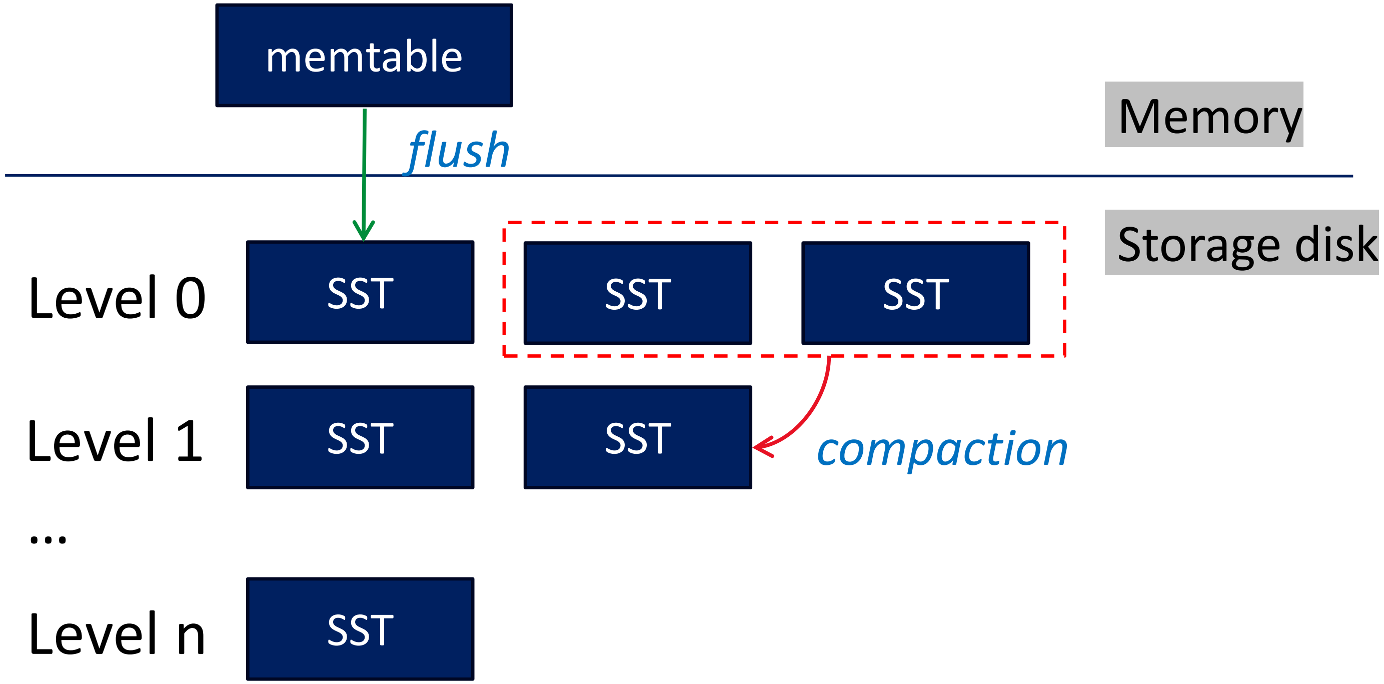
# Overview

1. **Introduction**
   - LSM-KVS
   - Tuning
2. **Related Work**
3. **Motivation**
   - Limitation
   - Prospect of LLMs
4. **Challenges**
5. **ELMo-Tune**
6. **Evaluation**
7. **Discussion and Future Work**

# Introduction – LSM-KVS

Log-Structured Merge-Key Value Stores (LSM-KVS) are an important cornerstone in the modern era. They have a high write throughput and demonstrate respectable read performance. This is achievable through their versatile design.

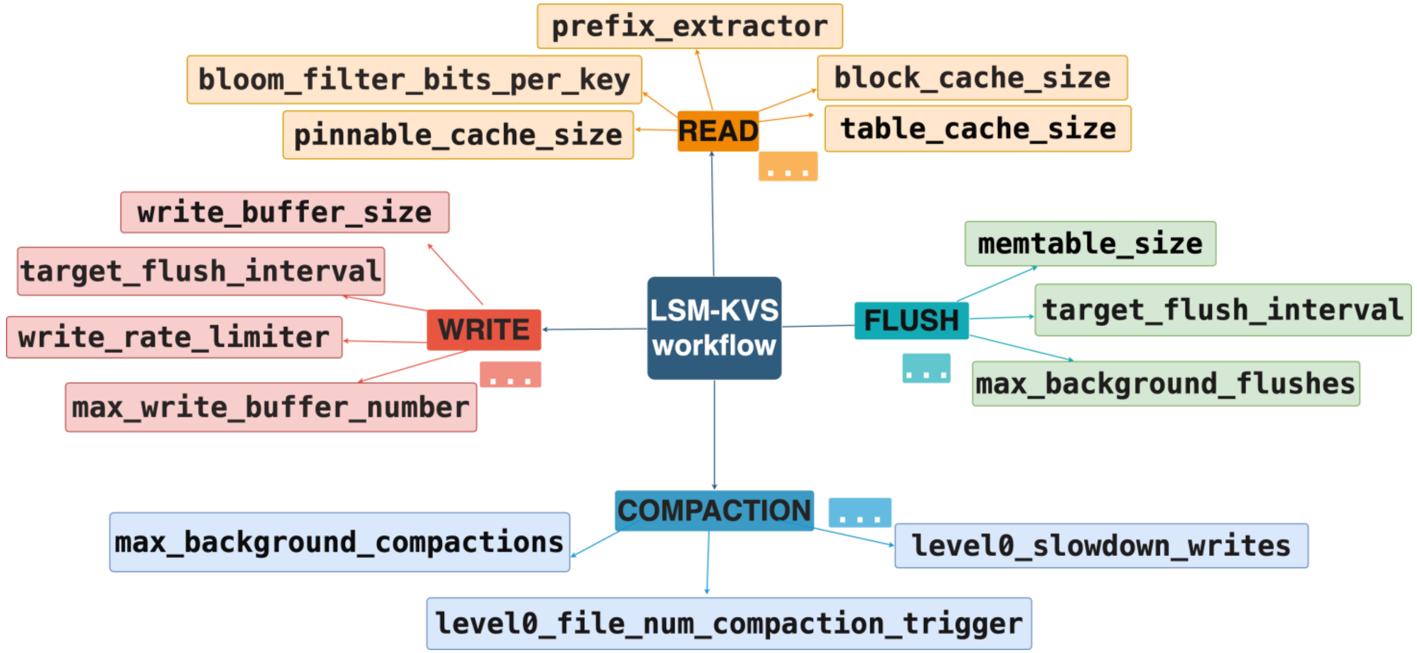
- Write Performance
  - Memtables
  - Flush
  - Append-only design
- Read Performance
  - Bloom Filters
  - Leveled Design
- Write Amplification
  - Compaction



# Introduction - Tuning

Different use cases demand for more specialized needs. To cater more specifically to everything,

- Optimization - Underlying codebase and configuration parameters are both modified to improve performance
- **Tuning - Exposed configuration parameters are tuned to improve performance**



# Related Work

**Manual Tuning**

- Flexible tuning of multiple parameters
- Educated initial values (based on system and workload configuration)
- Time consuming
- Necessity of a domain expert
- eg. Online Tuning Guides [1, 2]

**Auto Tuning**

- Subset of options (Bloom filters, memory buffers, compaction strategies)
- Trial and error (lack of understanding of system resources and workloads)
- Relatively quicker
- Plug and Play framework
- eg. Auto-Tuning Frameworks: RUSKey [3], Endure [4]

# Motivation - Limitation

## Manual Tuning

- ***Flexible tuning of multiple parameters***
- ***Educated initial values (based on system and workload configuration)***
- Time consuming
- Necessity of a domain expert

## Auto Tuning

- Subset of options (Bloom filters, memory buffers, compaction strategies)
- Trial and error (lack of understanding of system resources and workloads)
- ***Relatively quicker***
- ***Plug and Play framework***

# Motivation – Prospect of LLMs

## A New System that:

- Flexible for tuning of multiple parameters
- Educated initial values (based on system and workload configuration)
- Relatively quicker
- Plug and Play framework

can you generate an option file for RocksDB on HDD?

Certainly! Setting up RocksDB for optimal performance on HDD involves configuring several parameters in the options file (`Optionsfile.ini`). Below is a sample configuration that you can adjust based on your specific hardware and workload requirements:

ini Copy code

```ini
# Optionsfile.ini for RocksDB on HDD

# General Options
------------------
allow_mmap_reads = false          # Disable mmap reads
allow_mmap_writes = false         # Disable mmap writes
use_direct_reads = false          # Disable direct reads
use_direct_io_for_flush_and_compaction = false  # Disable direct I/O for flush and compact
max_open_files = 10000            # Adjust based on your system limits
bytes_per_sync = 524288           # Adjust based on your workload (recommendation: 524288 
base_background_compactions = 2   # Number of background compaction threads

# Block Based Table Options
---------------------------
block_size = 8192                # Typical HDD block size, adjust based on workload (recom
cache_index_and_filter_blocks = true  # Cache index and filter blocks
pin_l0_filter_and_index_blocks_in_cache = true  # Pin L0 filter and index blocks in cache
optimize_filters_for_hits = true  # Optimize Bloom filters for recent reads
whole_key_filtering = true       # Enable whole-key Bloom filter
```

## Modern Large Language Models

- LLMs are trained with opensource code, reports, papers. It can be considered an "expert" in some widely used opensource applications like RocksDB.
- LLMs have additional knowledge of system, how different systems can be set up, and different hardware setup and configurations

# Challenges

**Objective**

Is it possible to build an interactive, automatic, adaptive, and efficient tuning framework based on LLM like GPT-4 for open-source LSM-KVS (e.g., RocksDB) to achieve reasonable performance improvement?

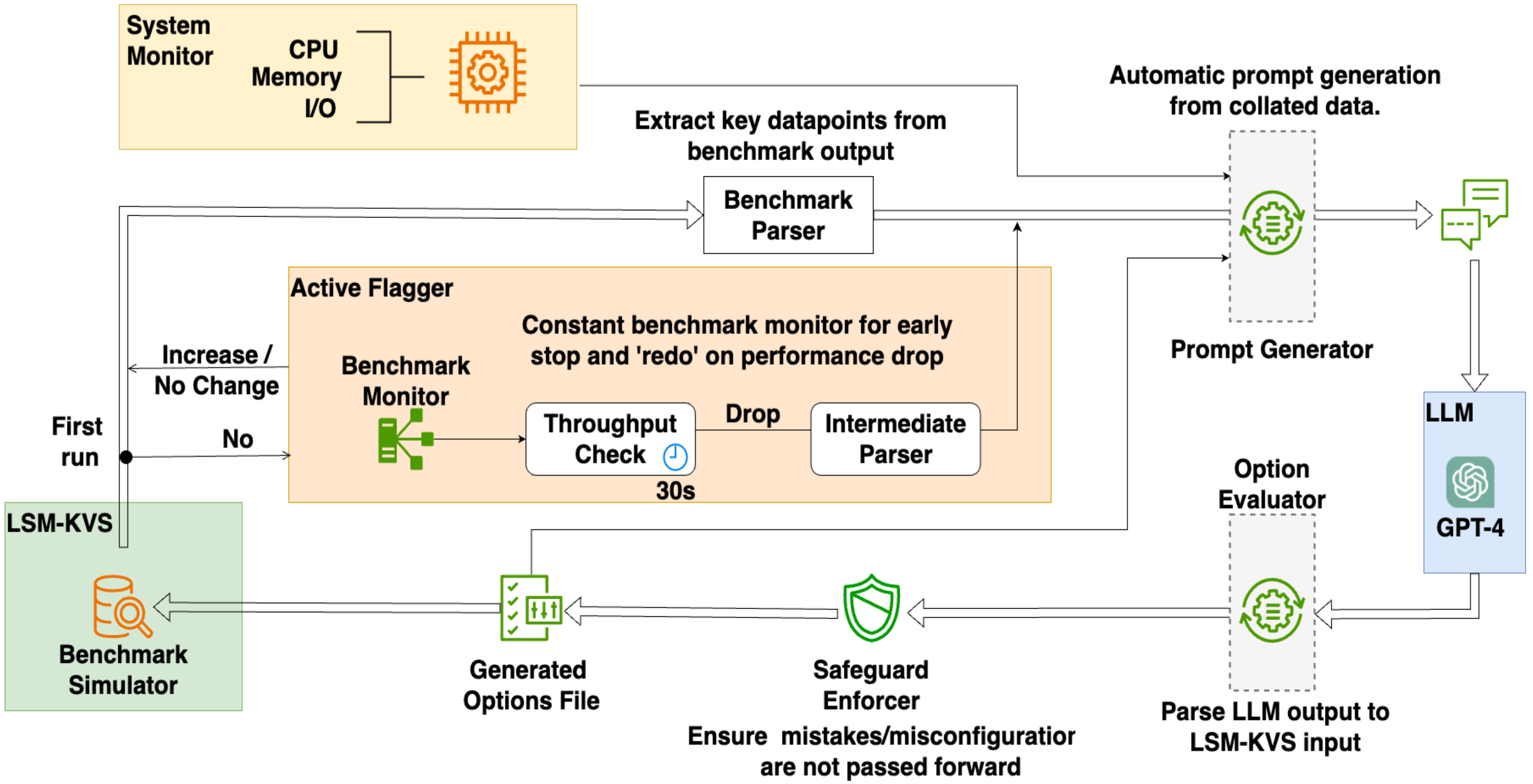
**Challenges**:

- How to integrate LLMs into a framework with LSM-KVS to achieve interactive auto-tuning.
- How to create tuning prompts that lead to desired output and optimize directions based on the hardware, software, workloads, and benchmarking results?
- How to process the benchmarking results and adopt LLM responses into a consolidated option file?
- How to mitigate LLM Hallucinations and Establish Safeguards for crucial parameters?

# ELMo-Tune

**Design**:

- Prompt Generation
- Option Evaluator
- Safeguard Enforcer
- Active Flagger

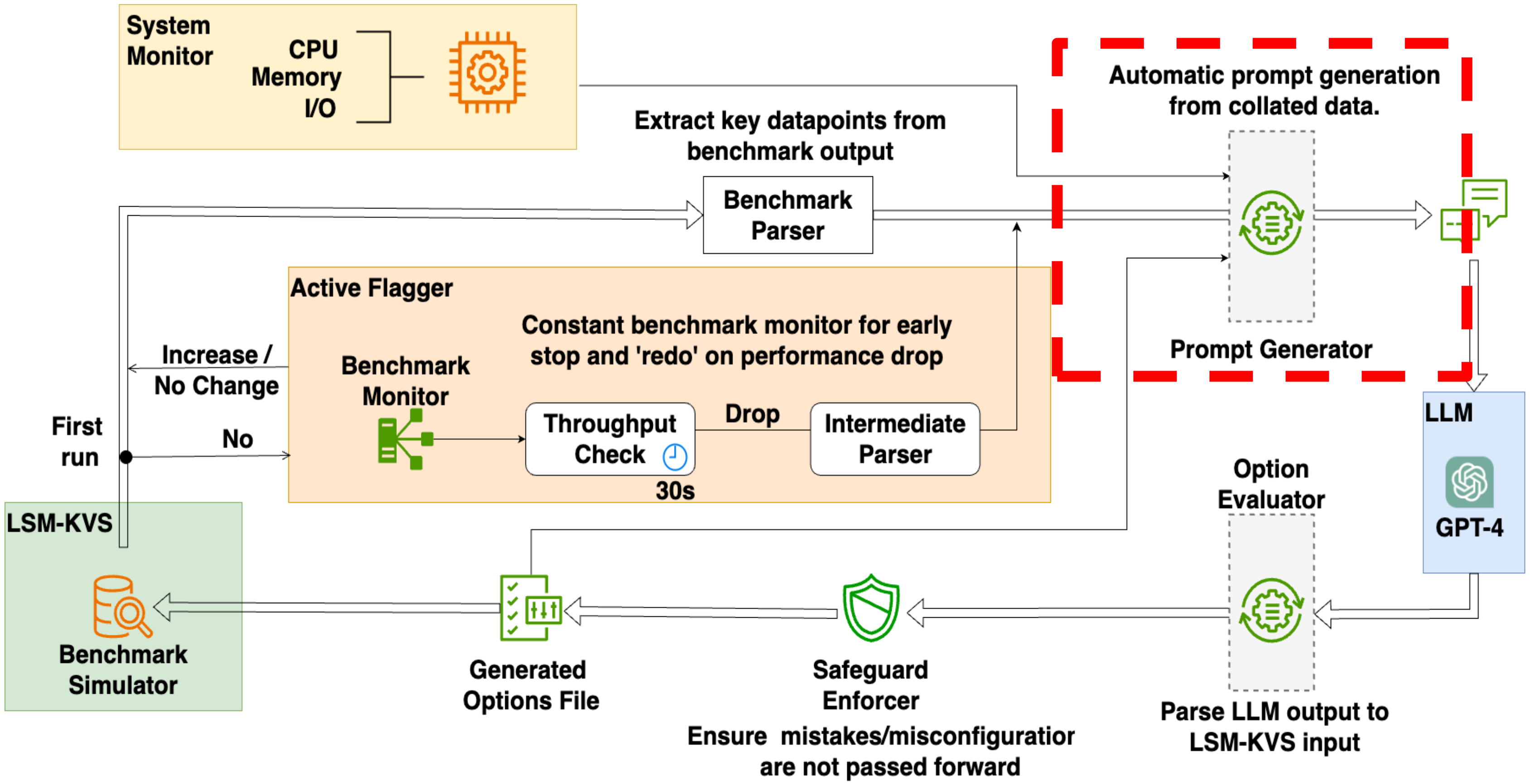


# ELMo-Tune

**Design:**

- **Prompt Generation**
  - *Combines runtime system information, benchmark results and options file*
  - *Sends data to LLM*

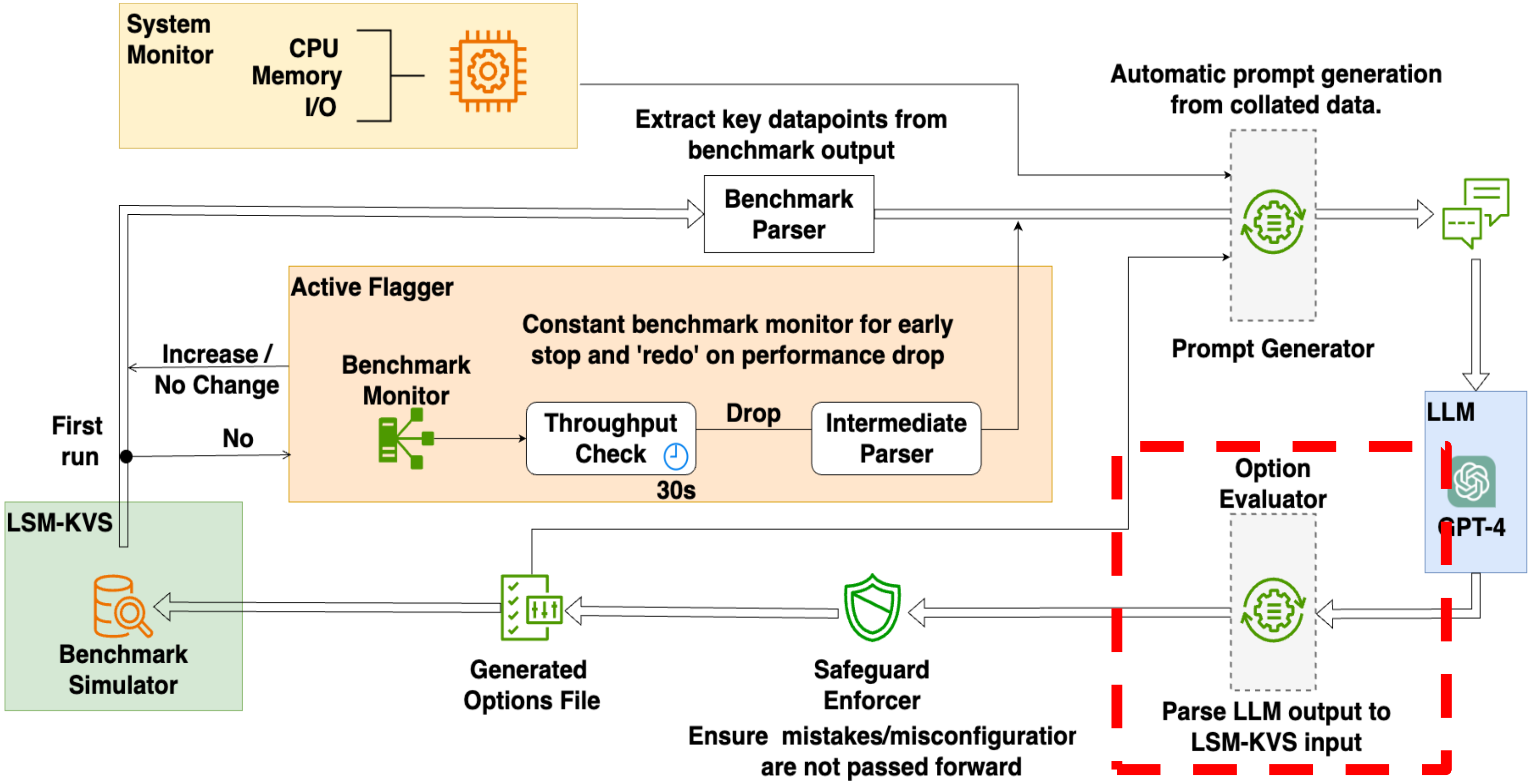
- Option Evaluator
- Safeguard Enforcer
- Active Flagger



# ELMo-Tune

**Design**:

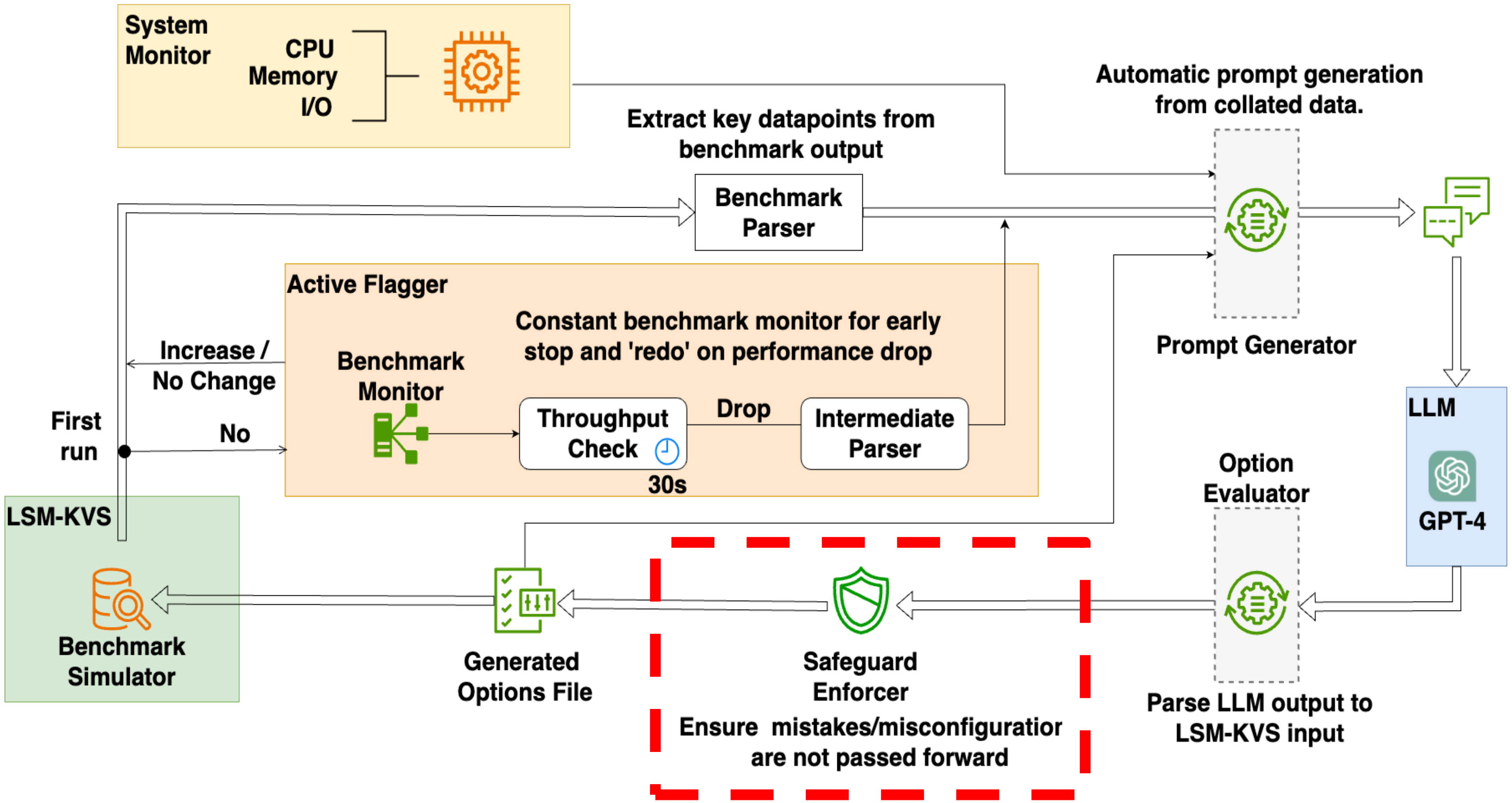
- Prompt Generation
- Option Evaluator
  - *Convert Natural Language to LSM-KVS understandable Options file*
  - *Log data to track changes made*
- Safeguard Enforcer
- Active Flagger



# ELMo-Tune

**Design**:

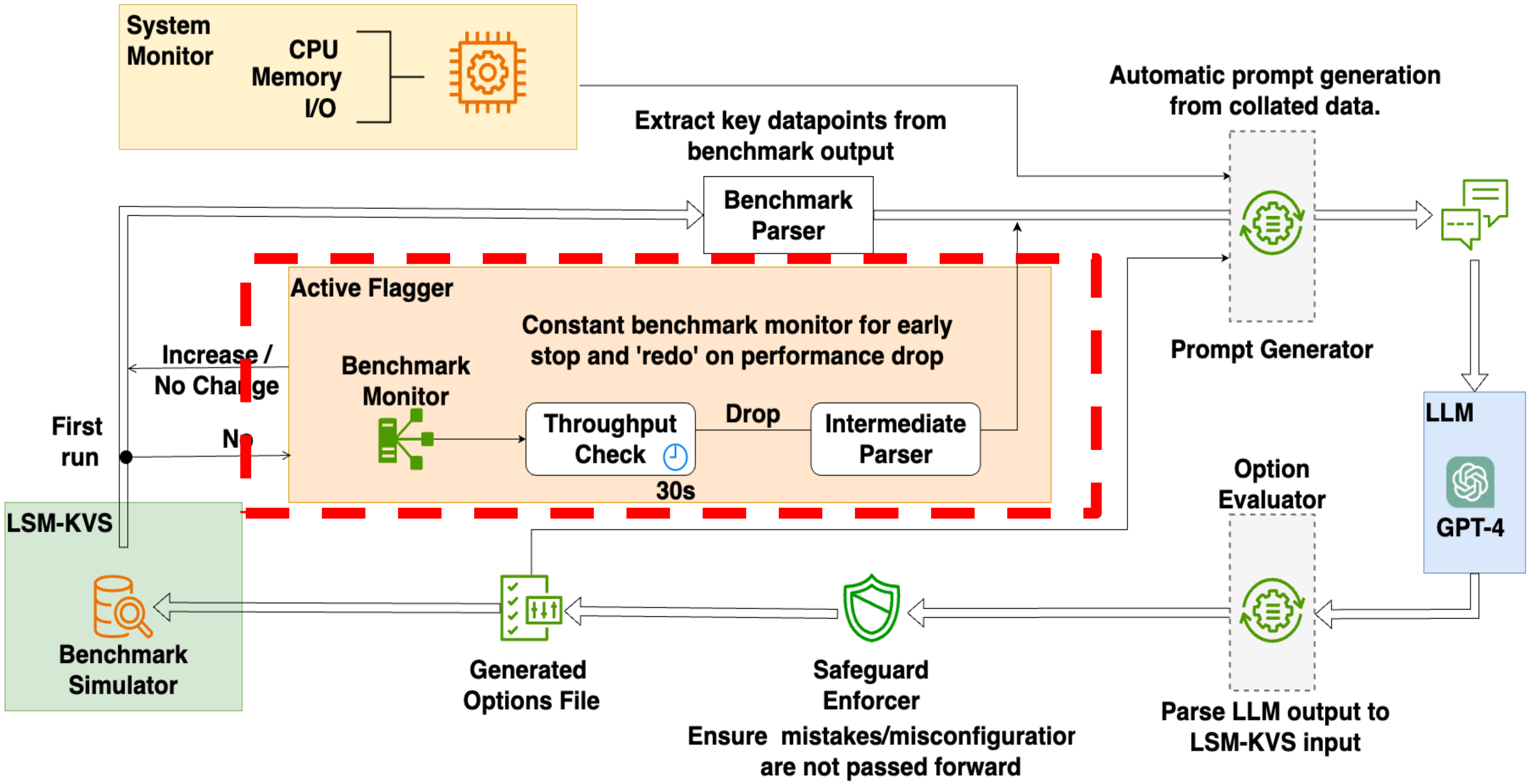
- Prompt Generation
- Option Evaluator
- Safeguard Enforcer
  - *Provide interface for user to disallow certain changes from LLM*
  - *Even if changed, values will be reset once through here.*
- Active Flagger



# ELMo-Tune

**Design**:

- Prompt Generation
- Option Evaluator
- Safeguard Enforcer
- Active Flagger
  - *Periodic notification on consecutive runs on performance degradation.*
  - *Compare with previous run (if any)*
  - *If degradation hits a threshold, reset with custom prompt explaining degradation.*

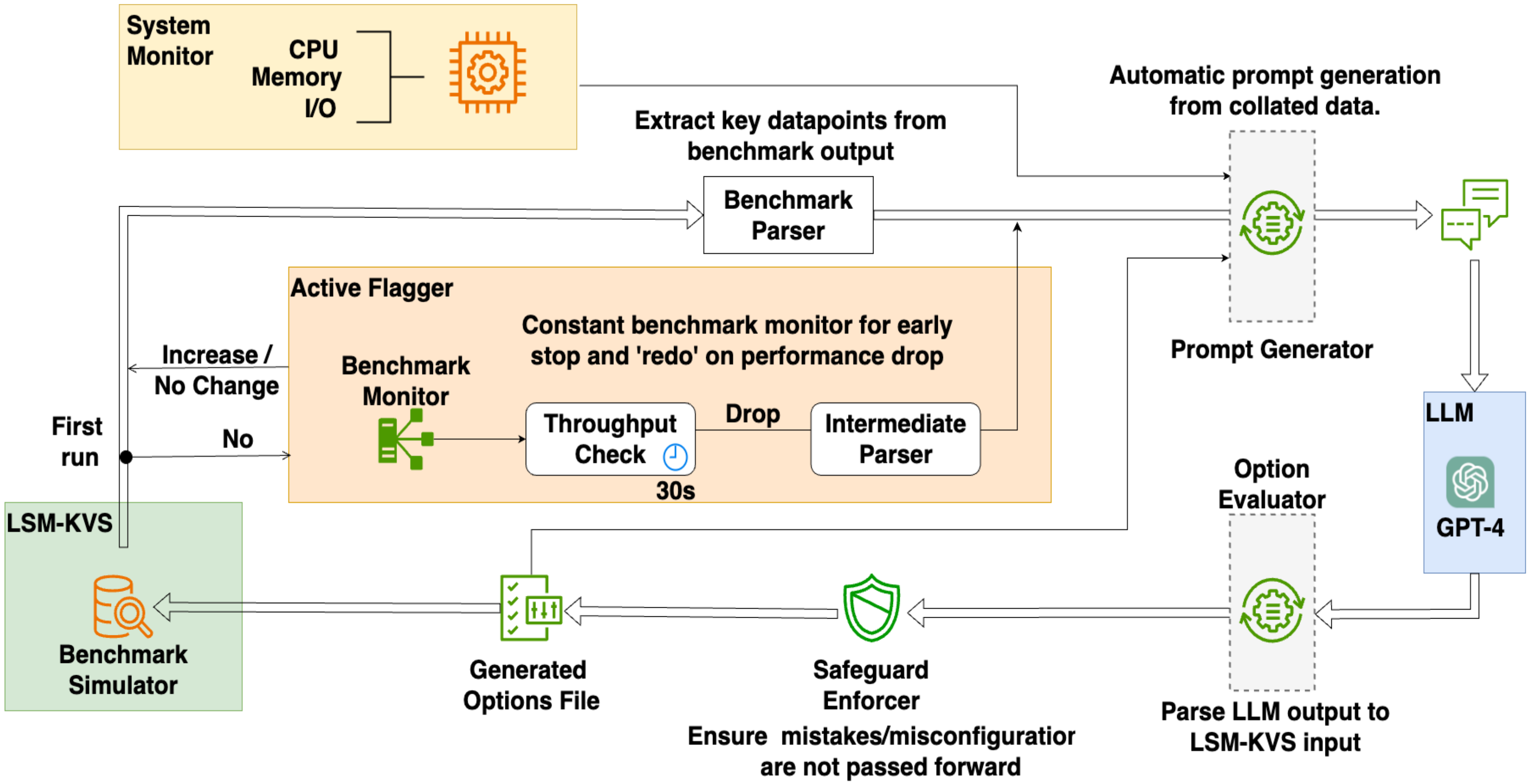


# ELMo-Tune

**Design**:

- Prompt Generation
- Option Evaluator
- Safeguard Enforcer
- Active Flagger

*Stoppage condition can be changed by the user. We find 7 iterations are enough to get to respectable improvements in results.*



# Evaluations

Evaluation conducted on a system with:

- LSM-KVS: RocksDB; Benchmark: db_bench; LLM: GPT-4
- Hardware: All combinations of (2 cores, 4 cores) CPU and (4GiB, 8GiB) Memory.
- Workloads: fillrandom, readrandom, mixgraph, readrandomwriterandom
- Storage Device: SATA HDD, NVMe SSD

Fillrandom on different hardware: improvement of 15.5% in throughput

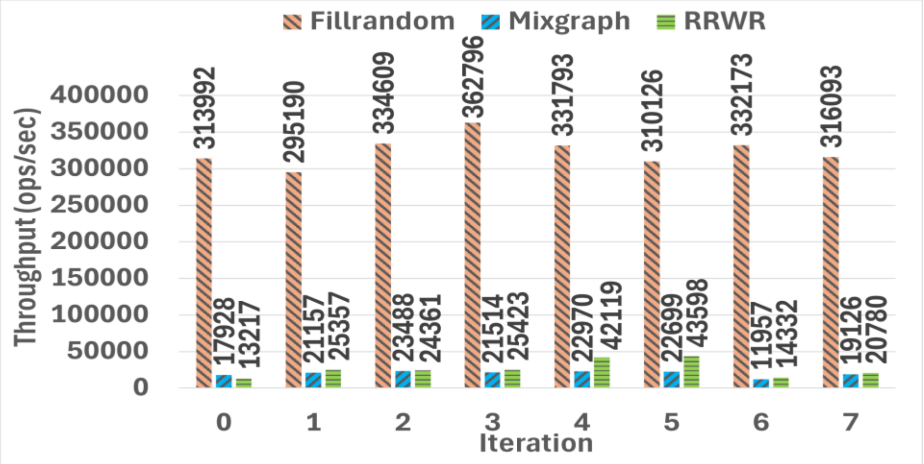
| | CPU (Cores) + Memory (GiB) Config | | | |
|---|---|---|---|---|
| | **2 + 4** | **2 + 8** | **4 + 4** | **4 + 8** |
| **Initial** | 320377 | 301677 | 313992 | 310574 |
| **Tuned** | 362460 | 348237 | 362796 | 329252 |

4+4 configuration on different workloads: 2X improvement in throughput

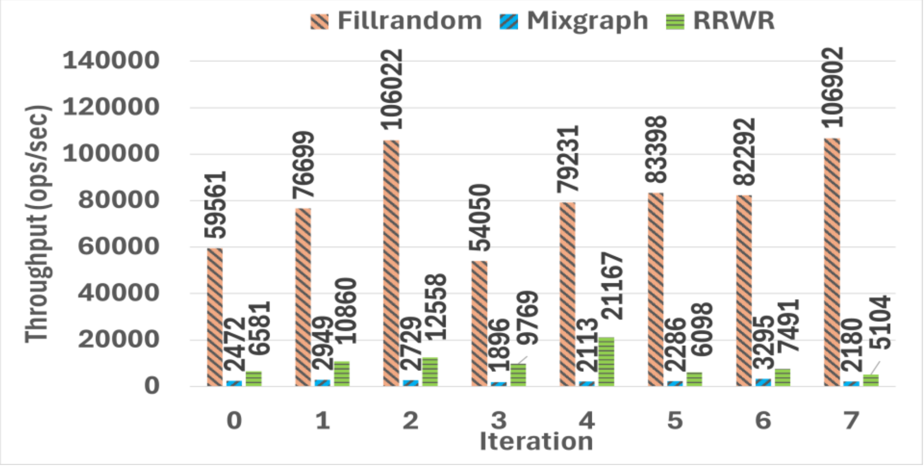
| | **FR** | **RR** | **RRWR** | **Mixgraph** |
|---|---|---|---|---|
| **Initial** | 313992 | 1928 | 13217 | 17928 |
| **Tuned** | 362796 | 5178 | 43598 | 23488 |

4+4 configuration on different workloads and storage devices: improvements on both devices – NVMe SSD + RRWR of ~3x; and SATA HDD + RRWR of ~3X.

NVMe SSD




SATA HDD



# Evaluations

Evaluation conducted on a system with:

- LSM-KVS: RocksDB; Benchmark: db_bench; LLM: GPT-4
- Hardware: All combinations of (2 cores, 4 cores) CPU and (4GiB, 8GiB) Memory.
- Workloads: fillrandom, readrandom, mixgraph, readrandomwriterandom
- Storage Device: SATA HDD, NVMe SSD

Fillrandom on different hardware: improvement of 15.5% in throughput

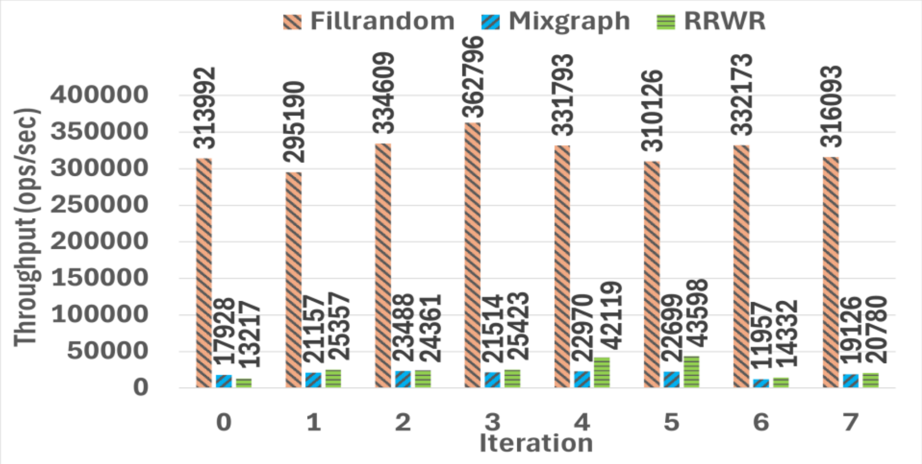
| | CPU (Cores) + Memory (GiB) Config | | | |
|---|---|---|---|---|
| | 2 + 4 | 2 + 8 | 4 + 4 | 4 + 8 |
| **Initial** | 320377 | 301677 | 313992 | 310574 |
| **Tuned** | 362460 | 348237 | 362796 | 329252 |

4+4 configuration on different workloads: 2X improvement in throughput

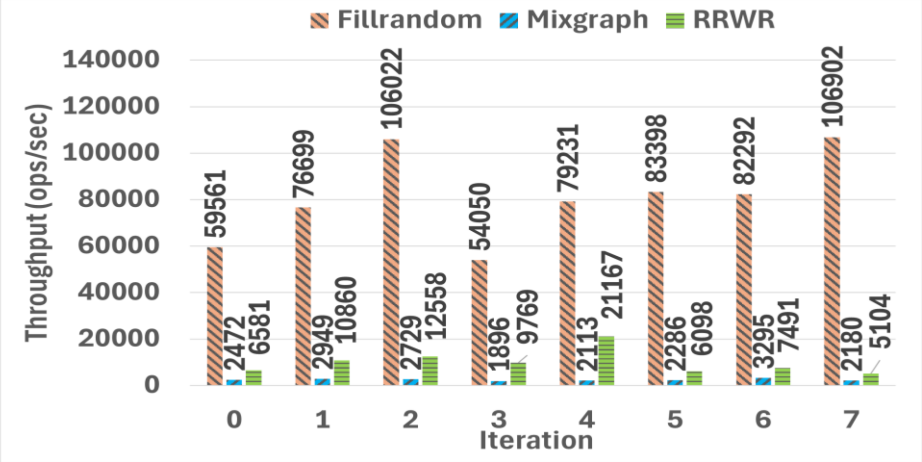
| | FR | RR | RRWR | Mixgraph |
|---|---|---|---|---|
| **Initial** | 313992 | 1928 | 13217 | 17928 |
| **Tuned** | 362796 | 5178 | 43598 | 23488 |

4+4 configuration on different workloads and storage devices: improvements on both devices – NVMe SSD + RRWR of ~3x; and SATA HDD + RRWR of ~3X.

NVMe SSD




SATA HDD



# Evaluations

Evaluation conducted on a system with:

- LSM-KVS: RocksDB; Benchmark: db_bench; LLM: GPT-4
- Hardware: All combinations of (2 cores, 4 cores) CPU and (4GiB, 8GiB) Memory.
- Workloads: fillrandom, readrandom, mixgraph, readrandomwriterandom
- Storage Device: SATA HDD, NVMe SSD

Fillrandom on different hardware: improvement of 15.5% in throughput

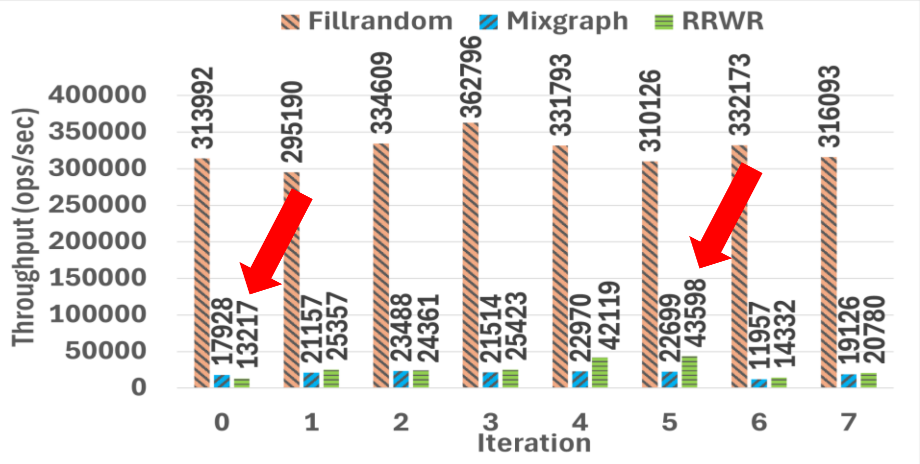
| | CPU (Cores) + Memory (GiB) Config | | | |
|---|---|---|---|---|
| | 2 + 4 | 2 + 8 | 4 + 4 | 4 + 8 |
| **Initial** | 320377 | 301677 | 313992 | 310574 |
| **Tuned** | 362460 | 348237 | 362796 | 329252 |

4+4 configuration on different workloads: 2X improvement in throughput

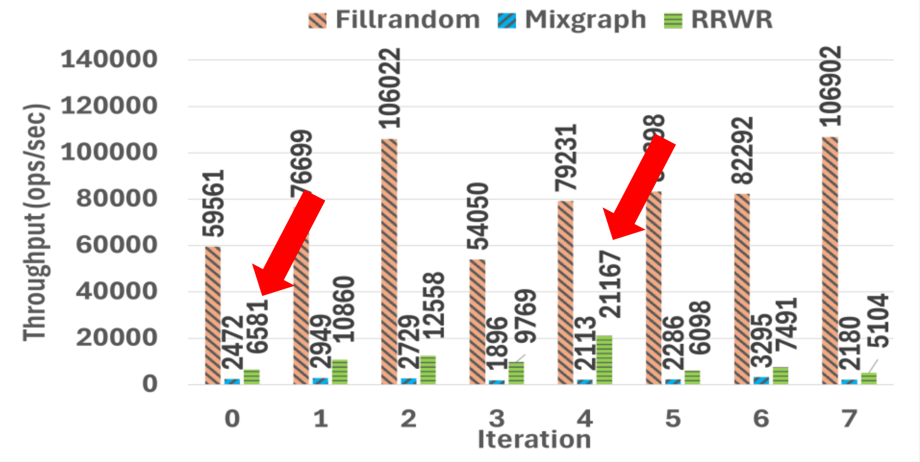
| | FR | RR | RRWR | Mixgraph |
|---|---|---|---|---|
| **Initial** | 313992 | 1928 | 13217 | 17928 |
| **Tuned** | 362796 | 5178 | 43598 | 23488 |

4+4 configuration on different workloads and storage devices: improvements on both devices – NVMe SSD + RRWR of ~3x; and SATA HDD + RRWR of ~3X.

NVMe SSD




SATA HDD



# Evaluations

Iteration-by-Iteration change for 2 CPU, 4GiB RAM, fillrandom configuration on HDD.

| Parameter | Original Value | 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 |
| **Result Throughput** | **42105** | **78969** | **83081** | **80963** | **82587** | **81157** | **91880** | **125019** |

Increased to 4MB from 2MB to improve sequential read performance during compaction, which is crucial for HDDs.

Reduced to align with Direct IO and system's I/O capabilities, optimizing read performance during compaction.

# Discussion and Future Plan

Framework can optimize LSM-KVS with different CPU, memory, and storage setups.

- ELMo-Tune adjusts ~10 options/iteration, after which there are marginal improvements.
- Performing iterations allows the LLM to experiment with past results
- The model responds in patterns similar to online blogs, preferring the same configuration options.

Future Research Plan

- Active Configuration – Remove necessity of Workload input
- Integration with Fine Tuning – Make the small changes too
- Broader testing – Add more test suits, and baselines
- Extension to other option depending systems (other databases and caching systems)

# References

[1] “RocksDB Tuning Guide.” *GitHub*, https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide. Accessed 7 July 2024.

[2] Ceph.Io — Ceph RocksDB Tuning Deep-Dive. 24 July 2022, https://ceph.io/en/news/blog/2022/rocksdb-tuning-deep-dive/.

[3] Mo, Dingheng, et al. “Learning to Optimize LSM-Trees: Towards A Reinforcement Learning Based Key-Value Store for Dynamic Workloads.” Proc. ACM Manag. Data, vol. 1, no. 3, Nov. 2023, p. 213:1-213:25. ACM Digital Library, https://doi.org/10.1145/3617333.

[4] Huynh, Andy, et al. “Endure: A Robust Tuning Paradigm for LSM Trees under Workload Uncertainty.” Proc. VLDB Endow., vol. 15, no. 8, Apr. 2022, pp. 1605–18. ACM Digital Library, https://doi.org/10.14778/3529337.3529345.

# Thank You!

# Q & A

ELMo-Tune

## Contact

Viraj Thakkar (viraj.dt@asu.edu)

https://asu-idi.github.io/contact/