ICMOS: Incremental Concept Mining for OS Kernel Configuration via LLMs Agentic Reasoning

Yang Han[1,2,3], Kaichun Yao[1], Libo Zhang[1]
[1]Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences, Hangzhou, China
[3]University of Chinese Academy of Sciences, Beijing, China
Email: hanyang23@mails.ucas.ac.cn, yaokaichun@outlook.com, libo@iscas.ac.cn

***Abstract*—Linux kernel configuration is critical for system performance, security, and adaptability. However, its vast configuration space, comprising over 17,000 configuration options, renders manual tuning both time-consuming and prone to errors. Existing methods largely rely on static heuristics or limited semantic rules, which struggle to capture complex configuration dependencies or adapt across diverse workloads. We introduce ICMOS, a framework that integrates large language models (LLMs) with a heterogeneous knowledge graph of kernel configuration concepts (OSKC-KG). By grounding LLM reasoning in structured semantics, ICMOS supports context-aware mining of configuration concepts and agentic concept evolution in response to new requirements and kernel updates. We evaluate ICMOS on configuration QA tasks and diverse real-world workloads, including databases, web servers, in-memory caches, and system benchmarks. ICMOS consistently outperforms LLM-only baselines, delivering higher accuracy, faster optimization, and robust system performance. Notably, it halves optimization time, reduces tail latency by 58.1%, and more than doubles configuration success rates. These results demonstrate that ICMOS provides a scalable and reliable framework for grounding LLM reasoning in structured semantics, thereby advancing kernel configuration understanding and optimization.**

***Index Terms*—LLMs, Knowledge Graph, OS kernel configuration, Agentic Reasoning.**

I. Introduction

Operating System Kernel Configuration (OSKC) refers to the process of selecting, understanding, and optimizing the configuration items that determine the kernel's behavior and structure. This process is critical to achieving system performance, security, and compatibility. However, modern Linux kernels comprise over 17,000 options with intricate dependencies, forming an intractably large configuration space that renders manual tuning infeasible. This complexity has spurred various automated solutions; however, existing approaches, such as Kmax [1], rely mainly on static heuristics or manual analysis, fail to capture semantic dependencies, and lack generalization across diverse hardware and workloads.

Although recent work has explored the use of large language models (LLMs), such as GPT-4 [2], and LLM-based frameworks like AutoOS [3] for kernel configuration optimization, several key challenges remain. First, configuration options often lack well-defined semantics, making it difficult for both human experts and LLMs to accurately infer their system-level implications. Second, LLMs are prone to hallucination [4], often producing invalid or suboptimal suggestions due to inadequate knowledge of the specific domain. Third, both rule-based and LLM-driven approaches exhibit significant inefficiencies. For instance, AutoOS demands 1 to 2 hours per optimization iteration and often fails to compile due to configuration conflicts.

To address these challenges, we propose **ICMOS**, a novel framework that synergizes LLMs with a heterogeneous knowledge graph for incremental concept mining in kernel configuration spaces. By combining the linguistic comprehension capabilities of LLMs with knowledge graphs' structured constraint modeling, ICMOS achieves configuration-aware reasoning, automatic semantic concept mining, and continuous evolution of mappings between configurations and concepts. Specifically, the foundation of ICMOS is **OSKC-KG**, a heterogeneous graph representation that systematically models kernel configuration spaces through: a *Configuration Graph* that models the hierarchical and dependency structures of kernel configuration items and a *Concept Taxonomy Graph* that captures the evolving functional semantics of configurations. Building upon this representation, ICMOS operates through a two-phase process: (1) **LLM-KG Synergy Mining**, Where the LLM traverses the configuration graph via structured context reasoning to mine functional semantics of configuration items and establish mappings to the concept taxonomy graph; (2) **Agentic Concept Evolution**, where the LLM agent dynamically incorporates user requirements or emerging kernel functions through concept-oriented agentic reasoning. Furthermore, our framework supports human-in-the-loop verification to ensure the correctness, interpretability, and functional relevance of mined concepts.

To validate ICMOS's ability to ground LLM reasoning in structured semantics, we further introduce a suite of downstream *OSKC Understanding Tasks*, including retrieval QA and performance tuning across both *system-level* metrics (e.g., CPU computation, file I/O) and *application-level* workloads (e.g., databases, web servers, in-memory caches). We evaluate ICMOS on these tasks and find that it consistently outperforms LLM-only baselines, achieving substantial gains in accuracy, system performance, and optimization efficiency. Notably, it reduces optimization time by nearly half, more than doubles

Corresponding author: libo@iscas.ac.cn

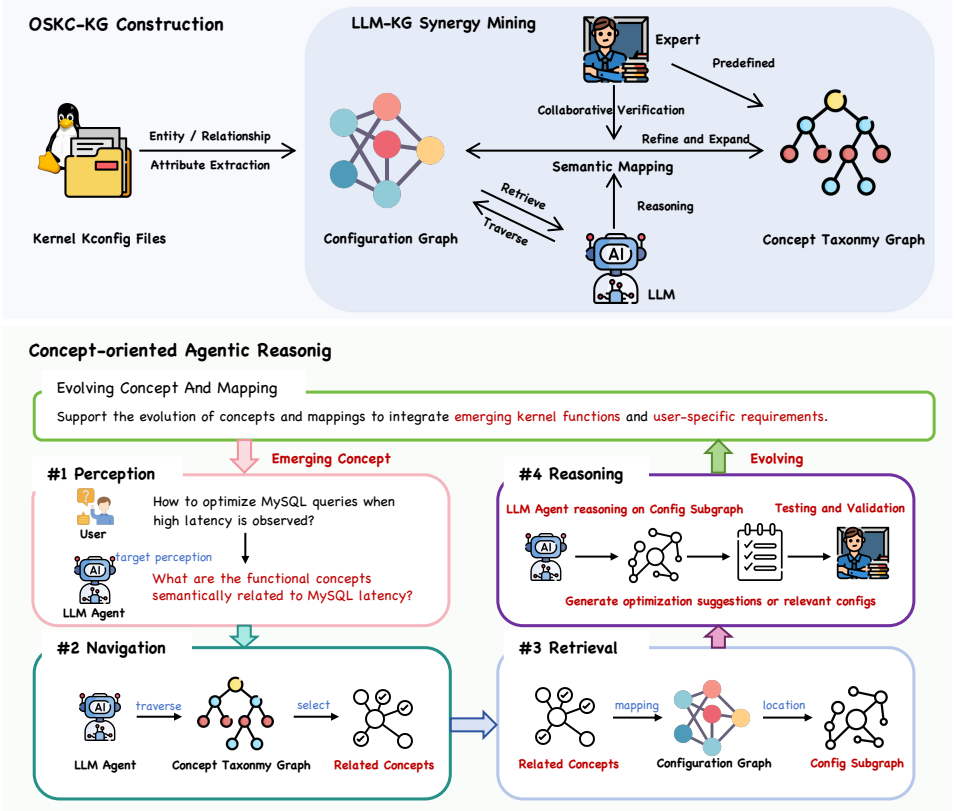

Fig. 1: Overview of ICMOS framework.

the configuration success rate, and cuts tail latency by up to 58.1% across representative workloads, underscoring its practicality and reliability.

To summarize our contributions:

1) We present **ICMOS**, the novel framework that integrates large language models with a heterogeneous knowledge graph (**OSKC-KG**) for kernel configuration. This enables structured, configuration-aware reasoning beyond heuristic or LLM-only approaches.
2) We introduce **incremental concept mining**, a novel approach that systematically extracts, organizes, and evolves the semantic concepts of kernel configuration options, establishing a dynamic and extensible concept taxonomy of OSKC semantics.
3) We design a two-phase process that combines *LLM-KG synergy mining* for context-aware semantic mapping and *agentic concept evolution* for adapting to user requirements and emerging kernel features, with human-in-the-loop verification for correctness and interpretability.
4) We establish a benchmark of two *OSKC understanding tasks* and empirically show that ICMOS consistently improves accuracy, efficiency, and performance over LLM-only baselines across realistic workloads.

II. The ICMOS Framework

We formalize the OSKC challenge as structuring configuration items, associating them with functional semantics, and enabling generalization to downstream tasks such as semantic QA and performance tuning. To address this, we propose **ICMOS**, an incremental concept mining framework that bridges raw kernel configuration options with high-level functional semantics through three synergistic components: (i) OSKC-KG, a heterogeneous knowledge graph unifying configuration structures and functional semantics; (ii) LLM-KG synergy concept mining, for automated, constraint-aware alignment of configuration options to semantic concepts; and (iii) Agentic concept evolution, enabling dynamic adaptation of semantics via concept-driven reasoning. The entire pipeline is reinforced by human-in-the-loop verification to ensure semantic correctness and interpretability (see Figure 1).

A. OSKC Concept Knowledge Graph

To bridge raw kernel configuration options with high-level functional semantics, we construct the *OSKC-KG*, a heterogeneous knowledge graph formally defined as $\mathcal{G}_{\text{OSKC}} = (\mathcal{V}, \mathcal{R}, \mathcal{T})$. It integrates two subgraphs:

a) Configuration Graph: The configuration graph captures structural and logical dependencies among Linux kernel options. Its nodes represent configuration entities (e.g., `menu`, `config`, `menuconfig`, `choice`), while its edges encode hierarchical structure (`SUB_OPTION`), dependency constraints (`DEPENDS`), and enforced implications (`SELECTS`). We automatically construct this graph by parsing the official Linux kernel's `Kconfig` files using `Kconfiglib` [5], yielding a

scalable representation of hierarchical and dependency structures across kernel versions.

b) Concept Taxonomy Graph: Modern kernel configuration options are often encoded in cryptic or hardware-specific identifiers, making semantic interpretation difficult for both human developers and LLMs. For example, the option `ATA_ACPI` appears obscure in isolation, yet its source description reveals associations with both *Power Management* and *Storage Support*. Such latent semantics are essential for managing and optimizing large-scale configuration profiles. To address this challenge, we construct a hierarchical *Concept Taxonomy Graph* that systematically organizes kernel configurations' functional semantics.

The taxonomy is seeded with expert-defined coarse-grained domains (e.g., *Security Features*, *Performance*, *Hardware Support*) and dynamically expanded via LLM-guided fine-grained concept mining (see following paragraph). This hybrid construction ensures both semantic coverage and extensibility, providing an interpretable space for aligning raw configuration items with high-level abstractions (An illustrative subset is shown in Table I). Crucially, OSKC-KG is not a simple juxtaposition of the two subgraphs. Instead, LLM-guided concept mining leverages structured information from the configuration graph to infer functional semantics, dynamically expanding the taxonomy and establishing cross-subgraph mapping relations. These mappings unify the two subgraphs into a coherent heterogeneous knowledge graph that supports interpretable reasoning and downstream tasks.

TABLE I: Part of Hierarchical Concept Taxonomy

Expert-defined Coarse Concepts		**LLM-mined**
Top-level	**Sub-level**	**Fine-grained Concepts**
Core Subsystem	Filesystem, Networking	Flash Filesystem, Wireless
Security Features	Access Control, Cryptography	Process Signal Control
Hardware Support	Storage, Peripheral	SCSI, Bus Support
Performance	Memory, I/O Optimization	PMD, DMA Optimization

c) LLM-KG Synergy Concept Mining: To bridge the configuration graph with the concept taxonomy graph and incrementally mine fine-grained functional semantics, we propose an LLM-KG synergy mining process. The core mechanism leverages a traversal of the configuration graph to extract rich contextual signals, including structural dependencies and natural language descriptions, and enables the LLM to predict appropriate semantic concepts. This process establishes interpretable mappings between configuration items and concepts while dynamically enriching the concept taxonomy.

We adopt a *Hierarchical Hybrid Traversal (HHT)* strategy to provide structure-aware context for the LLM. First, a global breadth-first search (BFS) identifies branching nodes (`menu`, `menuconfig`, `choice`) and assigns depth levels. Second, local depth-first search (DFS) collects descendant `config` nodes into *branch units*, enriched with two key contexts: (1) a *parent state* encoding high-level intent from ancestors, and (2) a *sibling state* aggregating peer semantics for contrastive reasoning. This structured context enables the LLM to produce coherent, taxonomy-aligned concept mappings.

The semantic mapping set is defined as:

$$\mathcal{M} \subseteq \mathcal{C}_{ext} \times \mathcal{V}_{cfg}, \tag{1}$$

where $\mathcal{C}_{ext}$ is the extended concept taxonomy (expert-defined and LLM-mined), and $\mathcal{V}_{cfg}$ is the set of configuration nodes. Each mapping is further annotated with a confidence score, rationale, and hierarchical insertion rule. For each configuration node $v \in \mathcal{V}_{cfg}$, we retrieve structured context from the configuration graph to guide the semantic alignment:

$$\mathcal{K}_v = \{\mathcal{P}_v, \mathcal{D}_v, P_\ell, S_\ell\}, \tag{2}$$

comprising: (1) the hierarchical path $\mathcal{P}_v$, (2) the configuration node description $\mathcal{D}_v$, (3) the parent state P_ℓ, and (4) the sibling state S_ℓ. This context enables the LLM to perform grounded, consistent reasoning by leveraging structural dependencies, mitigating hallucination and ambiguity.

Given the context $\mathcal{K}_v$, the LLM predicts a set of relevant concepts from $\mathcal{C}_{ext}$. The top predictions, filtered by a confidence threshold θ, are used to establish `RELATED_TO` mappings between the configuration item and its functional concepts. Simultaneously, any newly discovered concepts are integrated into the taxonomy via `SUB_CATEGORY` relations. All outputs support optional human-in-the-loop validation to ensure semantic accuracy and practical utility. This closed-loop process enables the OSKC-KG to evolve coherently, aligning low-level configurations with high-level semantics in a structured and interpretable manner.

B. Concept-oriented Agentic Reasoning

While the synergy mining process aligns configuration items with fine-grained concepts by leveraging the structured semantics of OSKC-KG, real-world requirements and kernel evolution continuously introduce new functionalities, such as workload-specific goals (e.g., MySQL latency optimization, secure container isolation) and kernel features (e.g., advanced scheduling or eBPF integration). To preserve semantic completeness and adaptability, ICMOS enables the incremental integration of emerging concepts into OSKC-KG.

We propose *Concept-oriented Agentic Reasoning*, a framework that leverages LLM agents to extend both the taxonomy $\mathcal{C}_{ext}$ and the semantic mappings $\mathcal{M}$. The process exploits three components of OSKC-KG: the configuration graph, the concept taxonomy, and their mappings, within a structured agentic pipeline.

Given a new concept C_{new}, the agent executes a structured four-stage pipeline: *perception, navigation, retrieval, and reasoning* (as illustrated in Figure 1). First, the agent reformulates C_{new} into a query prompt and performs guided traversal over $\mathcal{C}_{ext}$ to identify semantically related branches. The result is a candidate set of related concepts $\mathcal{T}_{\text{related}}$. Using the existing mapping $\mathcal{M}$, the agent retrieves the associated configurations:

$$\mathcal{V}_{\text{cand}} = \bigcup_{T_j \in \mathcal{T}_{\text{related}}} \{v \,|\, (v, T_j) \in \mathcal{M}\}, \tag{3}$$

which induces a subgraph $\mathcal{G}_{\text{config}} \subseteq \mathcal{V}_{cfg}$ representing the relevant configuration space.

The agent then performs retrieval-augmented reasoning over $\mathcal{G}_{\text{config}}$, leveraging metadata such as help texts, dependency constraints, and selection rules. This produces an increment of semantic mappings:

$$\mathcal{M}_{\text{new}} = \{(v, C_{\text{new}}, s, r) \mid v \in \mathcal{V}_{\text{cand}}\}, \tag{4}$$

where $s \in [0, 1]$ is a confidence score and r denotes the rationale. Finally, the integration of C_{new} into the taxonomy is guided by semantic alignment and subject to human-in-the-loop validation. If C_{new} is belongs to an existing branch, it is inserted via a `SUB_CATEGORY` relation; otherwise, it is added as a new root-level concept. The mapping set is then updated as $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_{\text{new}}$.

C. Human-in-the-Loop Verification

To ensure the reliability of the semantic mapping $\mathcal{M}$ and the evolving concept taxonomy $\mathcal{C}_{\text{ext}}$, we incorporate a human-in-the-loop verification mechanism that enables domain experts to examine the generated concept assignments by inspecting the associated confidence scores and reasoning traces, and subsequently validate their semantic correctness.

This verification process supports both global refinement of the taxonomy (e.g., merging overlapping concepts or adjusting hierarchical structures) and local validation of individual configuration-to-concept mappings. Furthermore, the refined concepts and mappings undergo *downstream task-driven validation* (Section III), where experts assess their semantic validity and effectiveness in real-world scenarios, such as configuration optimization and retrieval QA.

III. Experiment

To evaluate both the semantic quality of OSKC-KG and the practical utility of ICMOS, we establish a benchmark of two OSKC understanding tasks: *Configuration Retrieval QA*, which retrieves relevant configuration items and their semantic explanations for a natural-language query, and *Intelligent Configuration Optimization*, which generates optimized configuration sets for performance targets under kernel dependency constraints. We compare ICMOS against LLM-only and heuristic baselines. We first present the experimental setup and OSKC-KG characterization, followed by results on these two tasks.

A. Experimental Setup and Metrics

a) Setup: We construct OSKC-KG from the Linux 6.x kernel series (versions 6.1–6.15) and implement ICMOS with *DeepSeek-V3-0324* [6] as the core LLM. All data are stored in *Neo4j* to support efficient traversal, retrieval, and expansion. Optimization experiments are conducted in a controlled Linux environment running Ubuntu 24.04 (Linux 6.8, `aarch64`) with 4 CPU cores and 4 GB RAM, ensuring that all methods are evaluated under identical software and hardware configurations.

b) Metrics: We adopt task-specific metrics: (i) *Configuration Retrieval QA*: tag-level **Recall**, **F1**, **Top-3 Accuracy**, and option-level **Exact Match Rate**. (ii) *Intelligent Configuration Optimization*: **UnixBench** [7] scores (CPU, memory, I/O, scheduling) for system-level evaluation, and application-level **throughput** and **latency** (e.g., Apache, MySQL, Redis), measured with ApacheBench [8], sysbench [9], and memtier_benchmark [10]. We further report **tuning efficiency** (optimization time per cycle) and **conflict-free success rate**.

B. OSKC-KG Characterization

TABLE II: OSKC-KG Scale and Coverage Statistics

Type	Count	%
Nodes (Total: 18,927)		
`Config` / `Menuconfig`	17,242	91.3
`Concept` Nodes	1,303	6.8
`Menu` / `Choice` Nodes	382	1.9
Edges (Total: 145,191)		
`DEPENDS`	47,244	32.6
`SELECTS`	13,919	9.6
`SUB_OPTION`	17,455	12.0
`SUB_CATEGORY`	1,370	0.9
`RELATED_TO`	65,203	44.9
Mapping		
Mapped Configs/Menuconfigs	16,630 / 17,242	96.5
Avg. Tags per Config	3.9	–

The constructed OSKC-KG contains 18,927 nodes and 145,191 edges, including 17,242 configuration/menuconfig nodes and 1,303 concept nodes. Overall, 96.5% of configurations are semantically mapped, with an average of 3.9 tags per configuration (Table II). The initial end-to-end construction of OSKC-KG takes about 25 hours, including graph construction, concept mining, and expert validation. This cost is largely one-time; subsequent updates for new kernel versions or requirements are incremental and incur much lower overhead.

C. Configuration Retrieval QA

TABLE III: Kernel Configuration QA Results

Metric	LLM-only	LLM + OSKC-KG
Concept Understanding		
Tag Recall	66.7%	**88.9%**
Tag F1	49.2%	**65.4%**
Top-3 Accuracy	46.3%	**75.9%**
Configuration Selection		
Exact Match Rate	58.7%	**86.9%**
Span-level F1	85.9%	**96.0%**

To evaluate whether OSKC-KG enhances semantic retrieval for kernel configurations, we construct a curated evaluation set of 100 expert-annotated QA pairs, covering two key tasks: (i) *Concept Understanding*, which requires identifying the functional concepts associated with a given kernel configuration (e.g., mapping `CONFIG_CFS_BANDWIDTH` to concepts related to scheduling and resource control), and (ii) *Configuration Selection*, which asks the model to select relevant

configuration options given a functional requirement (e.g., identifying options related to namespaces for containerization).

As shown in Table III, OSKC-KG augmentation substantially outperforms the LLM-only baseline: concept recall increases from 66.7% to 88.9%, concept F1 improves by 16.2 percentage points (from 49.2% to 65.4%), and exact match accuracy for configuration selection rises from 58.7% to 86.9%. These gains demonstrate that structured semantic grounding not only improves retrieval fidelity but also enhances interpretability by aligning model outputs with domain concepts.

D. Intelligent Configuration Optimization

We evaluate ICMOS in practical, deployment-oriented settings, examining both performance gains and reliability under kernel constraints. To ensure that the produced configurations are feasible and deployable, we augment LLM reasoning with a Conflict-Aware validation mechanism that enforces dependency, selection, default-value, and visibility constraints derived from OSKC-KG, thereby preventing infeasible or mutually conflicting option sets.

We assess optimization along two complementary dimensions: **(i) System-level performance**, measuring CPU, memory, I/O, and scheduling efficiency using UnixBench; and **(ii) Application-specific tuning**, evaluating end-to-end performance for representative workloads (web servers, databases, in-memory caches) under high-concurrency benchmarks. This separation enables us to capture both micro-level efficiency and macro-level workload adaptability of the resulting configurations.

a) Comparison Setup: We compare three strategies: (i) *Default*: the official Linux configuration; (ii) *AutoOS*: LLM-only optimization without external knowledge; (iii) *ICMOS*(ours): the full concept-oriented agentic reasoning pipeline (Section II-B), augmented with OSKC-KG retrieval and a conflict-aware validation mechanism to ensure semantic grounding and feasibility. All evaluations follow the metrics defined in Section III-A.

TABLE IV: System-Level Performance on UnixBench

Test Case	Baseline	AutoOS	ICMOS
Dhrystone 2	8335.2	8391.1 (+0.7%)	**8449.8 (+1.4%)**
Whetstone	1586.3	1611.7 (+1.6%)	**1626.9 (+2.6%)**
Execl Throughput	2247.9	2389.8 (+6.3%)	**2401.0 (+6.8%)**
File Copy 1024b	4873.6	5121.4 (+5.1%)	**5177.6 (+6.2%)**
File Copy 256b	3279.1	3339.5 (+1.8%)	**3384.3 (+3.2%)**
File Copy 4096b	8164.2	9231.4 (+13.1%)	**10152.8 (+24.4%)**
Pipe Throughput	2177.4	1963.4 (-9.8%)	**2302.8 (+5.8%)**
Context Switch	78.8	86.7 (+10.0%)	84.4 (+7.1%)
Process Creation	733.0	810.1 (+10.5%)	**873.8 (+19.2%)**
Shell Scripts (1)	4826.4	5183.9 (+7.4%)	**5266.6 (+9.1%)**
Shell Scripts (8)	10161.7	9933.7 (-2.2%)	**10385.2 (+2.2%)**
System Call	1056.2	1093.3 (+3.5%)	**1099.7 (+4.1%)**
Index Score	2287.6	2348.4 (+2.7%)	**2432.1 (+6.3%)**

b) System-Level Performance: As shown in Table IV, ICMOS consistently improves all twelve UnixBench metrics, achieving +6.3% overall gains versus +2.7% for AutoOS, and up to +24.4% in file I/O and +19.2% in process creation. Unlike AutoOS, which occasionally introduces regressions due to uninformed configuration choices, ICMOS maintains stability by leveraging structural knowledge and conflict validation.

c) Application-Specific Tuning: Across six real-world workloads (Table VII), ICMOS yields substantial benefits, reducing tail latency by up to 58.1% and improving throughput by as much as 11.0%. For example, PostgreSQL achieves 26.4% lower P95 latency alongside notable QPS gains (+10.9%). In contrast, AutoOS often degrades performance (e.g., increased Nginx request time), highlighting the robustness of ICMOS across diverse workloads.

d) Efficiency and Reliability: As summarized in Table V, ICMOS reduces optimization time by 50% (39 min vs. 82 min per cycle) and increases configuration success rate from 29% to 76%, while nearly eliminating compilation/boot failures. These results confirm that conflict-aware reasoning not only enhances performance but also ensures reliable and deployable configurations.

TABLE V: Optimization Efficiency and Reliability

Metric	AutoOS	ICMOS	Imp. (%)
Average Optimization Time per Cycle	82 minutes	**39 minutes**	**47.6%**
Average Conflicts per Iteration	4.5	**1.4**	**68.9%**
Configuration Success Rate	29%	**76%**	**162.1%**

E. Ablation Study

TABLE VI: Concept Masking Ablation

Test Case	Baseline	AutoOS	ICMOS Masked	ICMOS
Dhrystone 2	8335.2	8391.1	8444.2	**8449.8**
Whetstone	1586.3	1611.7	1607.8	**1626.9**
Execl Throughput	2247.9	2389.8	2378.4	**2401.0**
File Copy 1024b	4873.6	5121.4	5120.6	**5177.6**
File Copy 256b	3279.1	3339.5	3342.7	**3384.3**
File Copy 4096b	8164.2	9231.4	8893.0	**10152.8**
Pipe Throughput	2177.4	1963.4	2200.8	**2302.8**
Context Switch	78.8	86.7	80.5	**84.4**
Process Creation	733.0	810.1	755.5	**873.8**
Shell Scripts (1)	4826.4	5183.9	4669.6	**5266.6**
Shell Scripts (8)	10161.7	9933.7	10268.3	**10385.2**
System Call	1056.2	1093.3	1063.7	**1099.7**
Index Score	2287.6	2348.4	2376.4	**2432.1**

To assess the role of taxonomy completeness, we conduct a concept-masking ablation by randomly removing 30% of mid- and leaf-level concepts, together with their edges and mappings, thereby reducing the semantic coverage available to the LLM agent during optimization. As shown in Table VI, ICMOS-Masked still outperforms the LLM-only AutoOS baseline, indicating that the overall reasoning framework remains robust even under incomplete semantic support. However, it consistently underperforms the full ICMOS model, especially on benchmarks that require finer-grained semantic distinctions. These results suggest that a more complete concept taxonomy provides stronger semantic guidance and leads to more effective optimization.

TABLE VII: Application-Specific Performance across Representative Workloads

Metric	Apache			Nginx		
	Baseline	AutoOS (%)	ICMOS (%)	Baseline	AutoOS (%)	ICMOS (%)
Time Taken for Tests (s)	0.235	0.232,+1.3%	**0.228,+3.0%**	0.268	0.257,+4.1%	**0.253,+5.6%**
Requests per Second (req/s)	42,490	43,095,+1.4%	**43,816,+3.1%**	37,324	38,916,+4.3%	**39,449,+5.7%**
Time per Request (ms)	2.354	2.320,+1.4%	**2.282,+3.1%**	2.679	2.570,+4.1%	**2.534,+5.4%**
Transfer Rate (KB/sec)	454,155	460,614,+1.4%	**468,332,+3.1%**	397,881	413,555,+3.9%	**420,533,+5.7%**
Max Request Time (Total, ms)	13	19,-46.2%	17.5,-34.6%	12	20,-66.7%	**11,+8.3%**
Metric	**MySQL**			**PostgreSQL**		
	Baseline	AutoOS (%)	ICMOS (%)	Baseline	AutoOS (%)	ICMOS (%)
Transactions per Second (TPS)	1,919.5	1,952.2,+1.7%	**1,996.9,+4.0%**	556.2	560.4,+0.8%,	**617.1,+11.0%**
Queries per Second (QPS)	39,502	40,191,+1.7%	**41,058,+3.9%**	11,489	11,570,0.7%	**12,744,+10.9%**
Average Latency (ms)	52.08	51.20,+1.7%	**50.06,+3.9%**	53.92	53.71,+0.4%	**47.96,+11.1%**
95th Percentile Latency (ms)	111.67	112.00,+0.3%	**104.84,+6.1%**	24.83	22.28,+10.3%	**18.28,+26.4%**
Metric	**Redis**			**Memcached**		
	Baseline	AutoOS (%)	ICMOS (%)	Baseline	AutoOS (%)	ICMOS (%)
Total Throughput (ops/sec)	257,552	267,080,+3.7%	**270,841,+4.0%**	447,110	461,583,+3.2%	**484,847,+8.4%**
Average Latency (ms)	1.56	1.50,+3.7%	**1.48,+4.6%**	0.90	0.88,+2.8%	**0.88,+2.9%**
P50 Latency (ms)	1.46	1.40,+4.2%	1.49,-1.1%	0.57	0.60,-4.7%	**0.52,+8.4%**
P99 Latency (ms)	3.31	3.28,-1.0%	**3.07,+7.3%**	3.97	3.73,+5.9%	3.96,+0.1%
P99.9 Latency (ms)	11.92	18.18,-52.4%	**4.99,+58.1%**	5.92	6.82,-5.1%	6.91,-16.8%
Throughput (KB/sec)	74,833	77,601 (-3.7%)	**78,694,+5.2%**	130,286	134,504,+3.2%	**137,616,+5.6%**

IV. Related Work

Early OS kernel configuration methods mainly rely on static analysis and heuristics over Kconfig files, such as Kmax [1], while learning-based approaches including VConf [11], DeepPerf [12], and transfer-based methods [13] improve automation but remain data-intensive and limited in semantic interpretability. Recent LLM-based systems such as AutoOS [3] represent an important early step toward kernel tuning with LLMs, but still lack deep semantic grounding of configuration options, leading to heuristic decision patterns, hallucination risks, and limited interpretability. More broadly, structured-knowledge and agentic approaches, including RAG [14], GraphRAG [15], ReAct [16], and SWE-agent [17], demonstrate the value of retrieval- and tool-augmented reasoning, but have rarely been applied to system-level kernel configuration.

V. Conclusion

We presented ICMOS, a novel framework that grounds LLM reasoning in the structured semantics of OSKC-KG to enable scalable, interpretable kernel configuration. By unifying configuration structure with dynamic concept taxonomies, ICMOS supports incremental concept mining and agentic evolution. Experiments demonstrate consistent gains over LLM-only baselines in both retrieval and optimization tasks, improving semantic fidelity, system performance, and reliability. Looking ahead, we will extend ICMOS to broader kernel versions and heterogeneous environments, further advancing knowledge-driven system automation.

Acknowledgment

This work was supported by the Code Semantic Understanding and Intelligent Generation project under Grant No. E0YD310101. AI tools were used only for language refinement.

References

[1] P. Gazzillo, "kmax," 2019.
[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
[3] H. Chen, Y. Wen, L. Cheng, S. Kuang, Y. Liu, W. Li, L. Li, R. Zhang, X. Song, W. Li *et al.*, "Autoos: make your os more powerful by exploiting large language models," in *Forty-first International Conference on Machine Learning*, 2024.
[4] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is inevitable: An innate limitation of large language models," *arXiv preprint arXiv:2401.11817*, 2024.
[5] U. E. Berg *et al.*, "Kconfiglib: A flexible python kconfig implementation," https://github.com/ulfalizer/Kconfiglib, 2018.
[6] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.
[7] K. D. Lucas, "UnixBench," 2015, version 5.1.2.
[8] Apache Software Foundation, *ApacheBench*, 2013, version 2.3.
[9] A. Kopytov, "Sysbench," 2023, version 1.0.20.
[10] Redis Labs, *memtier benchmark*, 2024, version 2.1.4.
[11] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: a reinforcement learning approach to virtual machines auto-configuration," in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 137–146.
[12] H. Ha and H. Zhang, "Deepperf: performance prediction for configurable software with deep sparse neural network," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1095–1106.
[13] B. Herzog, F. Hügel, S. Reif, T. Hönig, and W. Schröder-Preikschat, "Automated selection of energy-efficient operating system configurations," in *Proceedings Of The Twelfth ACM International Conference On Future Energy Systems*, 2021, pp. 309–315.
[14] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.
[15] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang *et al.*, "Retrieval-augmented generation with graphs (graphrag)," *arXiv preprint arXiv:2501.00309*, 2024.
[16] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The eleventh international conference on learning representations*, 2022.
[17] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "Swe-agent: Agent-computer interfaces enable automated software engineering," *Advances in Neural Information Processing Systems*, vol. 37, pp. 50 528–50 652, 2024.