TOWARDS AGENT-FRIENDLY DOCUMENTATION FOR DATABASE TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

System-tuning documentation provides parameter recommendations but often leaves their deployment-specific validation implicit, limiting their relevance across environments. We present PERFEVOLVE, a framework that augments static tuning guidance with *executable procedural knowledge* for LLM-based tuning agents. Offline, PERFEVOLVE ranks parameters by sensitivity, screens candidate ranges, and identifies interacting groups, compiling the results into procedures with measurements, decision rules, and validation checks. At deployment, these procedures guide local experiments or supply empirical priors to existing tuners. On PostgreSQL 16 with TPC-C and TPC-H workloads, integrating PERFEVOLVE into GPTuner improves matched-environment performance by 1.4%–6.1%, increases the valid-trial rate from 68%–81% to 100%, and yields gains of 45.8%–58.9% under cross-hardware transfer. With E2ETune, the full procedural representation achieves a 35.2% improvement over default PostgreSQL. These results support combining executable procedures with empirical reference data to improve tuning reliability and adaptation across deployments. Code and artifacts are available at `https://anonymous.4open.science/r/PerfEvolve-DF10`.

1 INTRODUCTION

Modern systems expose hundreds of configurable parameters governing memory, I/O, and concurrency win (2026); fre (2026); PostgreSQL Global Development Group (2023); Oracle Corporation (2024); red (2026a); Meta Platforms, Inc. (2026a). Their optimal settings depend on the workload, hardware, and software version, making configuration search prohibitively expensive Kroth et al. (2025); Pavlo et al. (2017; 2021); Zhu et al. (2017). System *documentation* provides valuable tuning knowledge Van Aken et al. (2017); Zhang et al. (2019); Duan et al. (2009); The PostgreSQL Global Development Group (2026); mys (2026); red (2026b); Meta Platforms, Inc. (2026b), including parameter semantics, valid ranges, recommended values, and operational cautions accumulated by developers and practitioners Beyer et al. (2016); Chen et al. (2025); Kunjir & Babu (2020). Recent LLM-based tuners extract constraints and priors from this knowledge to guide optimization Lin et al. (2026); Giannakouris & Trummer (2025); Sun et al. (2025).

However, these methods assume that *documentation contains the knowledge required to specialize a configuration to a deployment*. Through our systematic analysis of various systems (Linux, PostgreSQL, MySQL, and Kafka) PostgreSQL Global Development Group (2023); Oracle Corporation (2024); Meta Platforms, Inc. (2026a); Linux Kernel Developers (2026); Apache Software Foundation (2026), we identify three deficiencies that hinder documentation's utility as a tuning oracle:

- *Stale assumptions:* Documentation can lag behind system evolution. PostgreSQL's `random_page_cost` still recommends 4.0 since version 7.4, calibrated for rotational HDD, yet the optimal value on SSDs is closer to 1.1 Postgresqlco.nf (2025); PGTune (2024).
- *Context missing:* Documentation specifies *what* a parameter does but not *when* it is optimal. PostgreSQL recommends `shared_buffers` = 25% RAM with no distinction between OLTP and OLAP, SSD and HDD, or different workloads Sharma et al. (2018); Li et al. (2025).
- *Correlation absence:* Documentation overlooks parameter correlation. Our analysis shows that the `shared_buffers` × `work_mem` correlation explains 39% of performance variance, a dependency entirely absent from the official documentation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Insight: document *process*, rather than only *outcomes*. These limitations reflect a shared gap: static recommendations record tuning *outcomes* while leaving the underlying *decision process* implicit. For example, "set `shared_buffers` to 25% of RAM" encodes a conclusion under implicit conditions and may become stale as the environment changes, whereas "measure buffer hit ratio under representative load and adjust `shared_buffers` across candidate values" specifies what to measure, which experiments to run, and how to revise the decision. This reframes documentation as guidance for *profiling* the target system: identifying informative measurements, structuring experiments, and providing decision rules that help agents adapt prior knowledge to new environments, while allowing thresholds and safe ranges to be recalibrated when necessary.

Proposal. We present PERFEVOLVE, a framework for constructing *executable procedural knowledge* for LLM-based tuning agents. Specifically, PERFEVOLVE separates knowledge construction from deployment into two phases: **In the offline phase**, it profiles representative workloads to rank parameters by sensitivity, estimate safe ranges, and construct an interaction graph for joint optimization. It then compiles these results into a directed acyclic graph of executable skills with actions, decision rules, preconditions, postconditions, and reference observations. **In the online phase**, an LLM agent executes this graph on the target deployment, validates the offline knowledge, and performs targeted rather than unconstrained search.

We evaluate PERFEVOLVE on PostgreSQL 16 with read-heavy and write-heavy TPC-C and TPC-H, using both a matched VM cluster and a high-end server. With the optimizer and trial budget held fixed, replacing GPTuner's Lao et al. (2024) documentation-derived search space with PERFEVOLVE's knowledge improves the matched-environment result by 1.4%–6.1% and raises the valid-trial rate from 68%–81% to 100%. Under cross-hardware transfer, it improves GPTuner by 45.8%–58.9%. Injecting the same knowledge into the independently designed E2ETune Huang et al. (2025) yields gains on all three server workloads and reaches 35.2% above default on TPC-H. Together, these results support a focused claim: procedural knowledge makes documentation-guided tuning safer and more adaptable, without constituting a universal optimizer.

Contributions. This paper makes the following contributions:

- We characterize three structural limitations of tuning documentation across four production systems and quantify their consequences through a PostgreSQL study covering sensitivity, workload dependence, and parameter interactions.
- We reformulate system tuning documentation as *procedural knowledge*, transforming static recommendations (what to set) into reproducible workflows (how to determine what to set), and design PERFEVOLVE to construct them using sensitivity-based dimensionality reduction and interaction-topology discovery.
- We evaluate PERFEVOLVE with two distinct LLM-based database tuners, three workloads, and cross-hardware settings, demonstrating our design can improves the performance of state-of-the-art tuning systems.

2 CHARACTERIZATION OF DOCUMENTATION GAPS

This section examines whether conventional documentation provides sufficient knowledge for deployment-specific tuning. We first identify recurring gaps across four production systems and then quantify their consequences in PostgreSQL.

2.1 DOCUMENTATION DEFICIENCIES ACROSS SYSTEMS

We examine the official documentation of four widely deployed systems, including Linux Linux Kernel Developers (2026), PostgreSQL PostgreSQL Global Development Group (2023), MySQL Oracle Corporation (2024), and Apache Kafka Apache Software Foundation (2026). Table 1 summarizes representative examples. Across these systems, we observe three recurring gaps.

Stale operational assumptions. Defaults and rules of thumb often outlive the conditions that motivated them. Storage-sensitive parameters are a prominent example: guidance calibrated for rotational disks may be inappropriate for SSD or NVMe devices. PostgreSQL's `random_page_cost` re-

Table 1: Representative gaps in system-tuning documentation. Documentation is valuable for semantics and starting points; the recurring omission is a deployment-specific procedure for validating or adapting them.

System / parameter	Documented guidance	Missing procedure	Tuning implication
Stale operational assumptions			
Linux `vm.dirty_ratio`	Fixed percentage controlling when dirty-page writeback is triggered.	No device- and workload-specific calibration workflow.	A value suited to slower storage can defer writeback and create bursty I/O on a different storage stack.
PostgreSQL `random_page_cost`	Default cost of 4.0, with qualitative advice to lower it when random access is cheap or cached.	No benchmark-based rule for selecting a value on the target deployment.	On our SSD VMs, setting 1.0 improves TPC-C-w by $1.4\times$ but degrades TPC-C-r by 24%.
Missing deployment context			
PostgreSQL `shared_buffers`	25% of RAM is a reasonable starting point for a dedicated server.	No adaptation rule for workload mix, storage, or competing memory demand.	A fixed allocation can displace the OS page cache or leave too little memory for query execution.
MySQL `innodb_buffer_pool_size`	Large buffer pools are recommended for dedicated database servers.	No general procedure for accounting for co-located services.	Blindly applying a dedicated-server ratio can create memory contention under co-tenancy.
Kafka `num.partitions`	Partition count is related qualitatively to throughput and consumer count.	No workload-driven rule combining skew, consumer parallelism, and overhead.	Too few partitions limit parallelism; too many add coordination overhead.
Missing parameter interactions			
PostgreSQL `work_mem` × `max_connections`	Each parameter is documented separately with memory-related cautions.	No executable check for their aggregate memory demand under concurrency.	Safe per-operation memory depends on the number of simultaneous operations.
Kafka partitions × consumers	Both mechanisms and their qualitative relationship are documented.	No joint experiment for matching partitioning to observed consumer behavior.	Misalignment can leave consumers idle or increase coordination without useful parallelism.

mains 4.0 by default, although the documentation notes that lower values may be appropriate when random access is cheap or data is cached PostgreSQL Global Development Group (2023). While the trade-off is explained, no procedure is provided for calibrating the value on a target deployment.

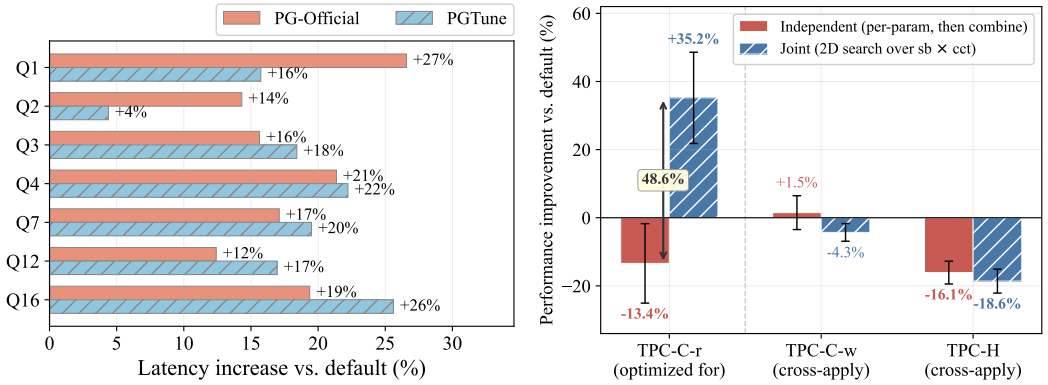
Missing deployment context. Documentation often provides defaults or starting points without workload- or deployment-specific decision rules. PostgreSQL, for example, recommends `shared_buffers`=25% of memory for a dedicated server PostgreSQL Global Development Group (2023), but does not specify how this value should vary with workload, storage, or competing memory demand. Similar gaps arise in MySQL buffer-pool sizing and Kafka partition counts.

Missing parameter interactions. Parameters are usually documented independently even when their underlying effects are coupled. For example, `work_mem` and `max_connections` jointly determine potential memory demand. Likewise, partition counts and consumer parallelism jointly determine Kafka's effective throughput. Documentation may mention such concerns qualitatively, but rarely supplies a test for deciding which parameters must be co-optimized.

These observations do not imply that documentation is incorrect. Rather, documentation is primarily descriptive: it explains parameter meaning and offers broadly useful starting points, but usually omits the experiments and decision rules needed to specialize those recommendations.

2.2 Case Study: PostgreSQL with 300+ Parameters

We use PostgreSQL because it exposes more than 300 parameters, has mature documentation, and is a common target of documentation-driven tuners Lao et al. (2024); Huang et al. (2025); Yang et al. (2025). Our PostgreSQL 16 testbed comprises 150 identically configured virtual machines, each with 2 vCPUs, 8 GB DRAM, and SSD storage. We study read-heavy TPC-C (TPC-C-r), write-heavy TPC-C (TPC-C-w), and TPC-H. Unless stated otherwise, we report performance relative to the default configuration and average three independent repetitions.



(a) Query-level regressions under static guidance.

(b) Cross-workload transfer of tuned configurations.

Figure 1: Consequences of static tuning. (a) PG-Official and PGTune increase latency on seven selected TPC-H queries. (b) Independent and joint tuning of `shared_buffers` (sb) and `checkpoint_completion_target` (cct) on TPC-C-r, cross-applied to TPC-C-w and TPC-H.

Finding 1: Static guidance can be locally harmful. We evaluate PostgreSQL's official guidance (PG-Official) and PGTune PGTune (2024). Figure 1a shows latency increases of 4–27% across seven TPC-H queries with large sorting or aggregation intermediates. Both allocate 2 GB to `shared_buffers` (25% of RAM) while leaving `work_mem` at 4 MB, reducing OS page cache without preventing disk spills. Checkpoint tuning shows a similar mismatch: although PostgreSQL 16 defaults to 0.9 and discourages smaller completion targets, our sensitivity scan finds that 0.5 improves TPC-C-r by 15% and 0.0 improves TPC-C-w by 24% on SSD-backed VMs. Thus, generally reasonable guidance can become harmful when detached from workload and hardware context.

Finding 2: Good configurations do not necessarily transfer across workloads. To isolate workload dependence, we optimize `shared_buffers` and `checkpoint_completion_target` on TPC-C-r and apply the resulting configuration unchanged to the other workloads. Figure 1b shows that the jointly tuned configuration gains 35.2% on TPC-C-r but loses 4.3% on TPC-C-w and 18.6% on TPC-H. This is not an optimization failure; it shows that an optimum is conditional on the workload that produced it. Documentation that records only selected values, without the workload context and a transfer test, cannot indicate when those values should be retained, revised, or discarded.

Finding 3: Pairwise interactions are frequent enough to require explicit screening. We apply the coarse stage of the two-stage procedure in §3.2 to 11 high-sensitivity parameters on TPC-C-r. Each of the $\binom{11}{2} = 55$ pairs is evaluated with a 2×2 factorial design using one repetition per cell. As Figure 2 shows, 33 pairs exceed the 15% candidate threshold, and eight of the ten highest-scoring pairs cross subsystem boundaries (Table 2). This coarse screen deliberately favors recall: its single-repetition design can overestimate interaction strength, so the 33 pairs are candidates rather than confirmed interactions. Nevertheless, their prevalence makes an independence assumption unsafe and motivates the replicated, FDR-corrected verification in Stage B before parameters are optimized separately.

Summary. These findings share a common root cause: documentation encodes *what to set*, but not *how to determine what to set*. Its recommendations may become stale across hardware, fail to capture workload dependence, and overlook parameter interactions. This motivates a better form of knowledge that encodes the *tuning process*, rather than only its results.

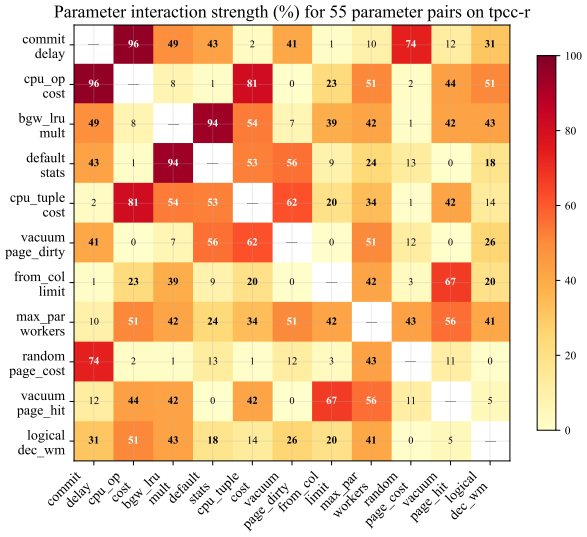


Figure 2: Coarse interaction-screening scores for 55 parameter pairs on TPC-C-r. Scores above 15% advance to Stage B verification.

Table 2: Ten largest coarse-screen scores. ✓ marks a cross-subsystem pair.

A	*B*	Score	X?
commit_delay	cpu_op_cost	**96.0**	✓
bgw_lru_mult	default_stats	**93.7**	✓
cpu_op_cost	cpu_tuple_cost	80.9	✗
commit_delay	random_page_cost	73.6	✓
from_col_limit	vac_page_hit	66.8	✓
cpu_tuple_cost	vac_page_dirty	62.0	✓
max_par_workers	vac_page_hit	56.0	✓
default_stats	vac_page_dirty	55.5	✓
bgw_lru_mult	cpu_tuple_cost	53.5	✓
cpu_tuple_cost	default_stats	52.6	✗

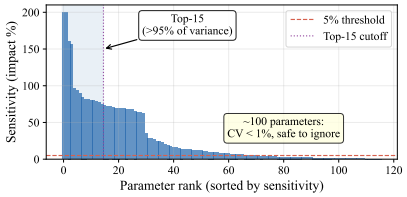


Figure 3: Parameter-sensitivity long tail on TPC-C-r (116 parameters; 15 above 5%).

3 PERFEVOLVE DESIGN

PERFEVOLVE converts offline profiles into procedural knowledge for LLM-based tuning agents (Figure 4). Offline, *dimensionality reduction* identifies influential parameters and candidate ranges, while *interaction topology discovery* identifies parameters for joint optimization. Online, an agent uses the resulting procedures to validate prior knowledge and guide deployment-specific tuning.

Design goals. The knowledge is *executable*, with concrete actions and decision rules; *verifiable*, with preconditions and postconditions that expose deviations from offline profiles; and *transferable*, separating reusable procedures from environment-specific reference data.

Workflow. Given n parameters and W representative workloads, PERFEVOLVE retains $k \ll n$ influential parameters, screens their $\binom{k}{2}$ pairs for interactions, groups them into connected components, and compiles them into a skill DAG (§3.3). At deployment, an agent validates offline assumptions and searches the prescribed parameters and groups, amortizing broad profiling across deployments.

3.1 DIMENSIONALITY REDUCTION

To reduce the large configuration space Kanellis et al. (2020); Ventura et al. (2021), we use a one-factor-at-a-time scan to rank parameters and characterize their candidate ranges.

Sensitivity scan. For each parameter x_i, we evaluate $L_i \in \{3, \ldots, 9\}$ values within its documented safe range, holding all other parameters at their defaults. Each configuration is measured $r = 3$ times per workload. We define sensitivity as the throughput range normalized by default throughput:

$$\mathrm{CV}_{i,w} = \frac{\max_v \overline{\mathrm{TPS}}(x_i{=}v,\, w) - \min_v \overline{\mathrm{TPS}}(x_i{=}v,\, w)}{\overline{\mathrm{TPS}}(x_i{=}\text{default},\, w)}, \tag{1}$$

where $\overline{\mathrm{TPS}}$ averages repetitions. We aggregate across workloads as $\mathrm{CV}_i = \max_w \mathrm{CV}_{i,w}$ and retain the top-k parameters above τ_s (5% by default). The scan requires $rW \sum_i L_i$ runs: about 6,297 for PostgreSQL with $n = 116$ and $W = 3$, which completes within one day on our 150-VM cluster.

Outputs. The scan produces global and workload-specific rankings Li et al. (2025), screened ranges excluding observed crashes or throughput losses above 50% relative to default, and response-shape labels (monotonic, non-monotonic, or threshold-like) to guide search. Measurements support deployment-time verification. In PostgreSQL, only 15 of 116 parameters exceed the 5% sensitivity threshold, while roughly 100 affect throughput by less than 1% (Figure 3).

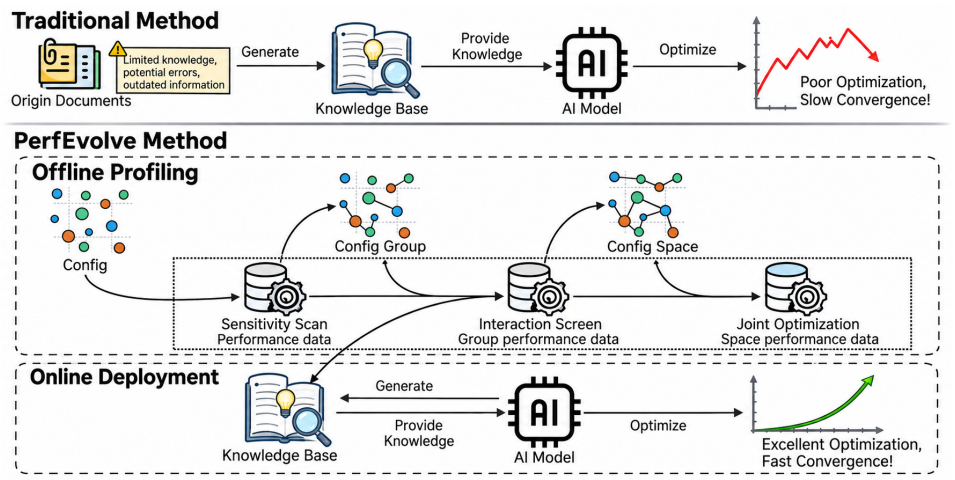


Figure 4: PERFEVOLVE workflow. Offline profiling identifies sensitive parameters, interactions, and joint-search groups; at deployment time, an LLM agent uses this knowledge for targeted tuning.

3.2 INTERACTION TOPOLOGY DISCOVERY

Individual sensitivity does not capture joint effects. We therefore use coarse factorial screening followed by replicated ANOVA Cohen (1988) to identify interacting parameter pairs.

Stage A: coarse screening. For each retained pair (x_i, x_j), we run a 2×2 factorial experiment using the extreme values observed during sensitivity scanning. With one observation per cell, we compute

$$\text{Int}\%_{ij} = \frac{|\overline{y}_{++} - \overline{y}_{+-} - \overline{y}_{-+} + \overline{y}_{--}|}{|\overline{y}_{++} + \overline{y}_{+-} + \overline{y}_{-+} + \overline{y}_{--}|/4}, \tag{2}$$

where $\overline{y}_{ab}$ denotes throughput at levels (a, b). Pairs above 15% advance to Stage B, while those below 5% are treated as independent. The permissive upper threshold favors recall because the unreplicated 2×2 design overestimates interaction strength by approximately $2\times$ in our validation.

Stage B: confirmation. Each candidate pair is evaluated using a 4×4 factorial design with $r = 3$ repetitions per cell (48 runs per workload). We quantify interaction strength as $\eta^2_{ij} = \frac{\text{SS}_{ij}}{\text{SS}_{\text{total}}}$, where SS_{ij} is the interaction sum of squares Lakens (2013). After Benjamini–Hochberg correction Benjamini & Hochberg (1995), we retain edges with $\eta^2_{ij} > 0.15$ and corrected $p < 0.05$.

Search decomposition. We construct a weighted undirected graph whose vertices are retained parameters and whose edges are confirmed interactions weighted by η^2_{ij}. Connected components define joint-search groups; isolated parameters are optimized individually. In PostgreSQL, the largest components contain three to four parameters. A four-level search over a three-parameter component requires $4^3 = 64$ configurations per workload, avoiding exhaustive enumeration of the full 4^{15} space.

3.3 PROCEDURAL DOCUMENT GENERATION AND DEPLOYMENT

Document generation. PERFEVOLVE compiles offline profiles into a procedural tuning document represented as a DAG of executable *skills*. Each skill specifies preconditions, measurements and actions, branching criteria, postconditions, and reference data. *Parameter skills* encode sensitivity, safe ranges, response shapes, and workload dependence; *component skills* define interaction-group search spaces and acceptance criteria; and an *orchestration skill* sequences verification, optimization, and cross-workload validation.

Deployment. PERFEVOLVE supports *full procedural execution* and *knowledge injection*. In full execution, an agent traverses the skill DAG, runs benchmarks, applies decision rules, and verifies postconditions. For knowledge injection, PERFEVOLVE exports the subset of procedural knowledge accepted by an existing framework. For GPTuner Lao et al. (2024), this includes the retained parameters, empirical safe ranges, and interaction annotations in JSON, allowing GPTuner to keep its optimizer while replacing documentation-derived priors with measured, version-specific knowledge.

4 EVALUATION

We address four research questions: **RQ1:** How does PERFEVOLVE affect end-to-end tuning performance (§4.1)? **RQ2:** How does procedural knowledge compare with declarative recommendations (§4.2)? **RQ3:** What are the cost and return of offline profiling (§4.3)? **RQ4:** How do parameter ranking, safe ranges, and interaction knowledge contribute to performance (§4.4)?

Experimental setup. The environment comprises 150 PostgreSQL 16 VMs (2 vCPU, 8 GB RAM, SSD), used for both offline profiling and deployment evaluation. For cross-hardware transfer, we use a 192-core Xeon 8575C server with 1 TB RAM and 8×H100 GPUs; the GPUs serve only E2ETune inference.

Table 3: Compared methods; PERFEVOLVE changes only the knowledge source.

Method	Type	# Par.
PG-Official / PGTune	Static rules	∼100
GPTuner	LLM + BO	57
GPTuner+PERFEVOLVE	LLM + BO	15
E2ETune	LLM one-shot	44
E2ETune+PERFEVOLVE	One-shot + prompt	44

Baselines. We evaluate two complementary LLM-based tuners (Table 3). **GPTuner** (Lao et al., 2024) extracts documentation constraints for the SMAC Bayesian optimizer (Hutter et al., 2011) (57 parameters; 30–100 trials), while **E2ETune** (Huang et al., 2025) fine-tunes Mistral-7B on historical data and emits a 44-parameter configuration in one inference. For each, we replace only the knowledge source with PERFEVOLVE while keeping the optimizer or model unchanged. Section 2.2 additionally evaluates PG-Official and PGTune PGTune (2024) as static baselines.

Workloads. Using BenchBase (Difallah et al., 2013), we evaluate **TPC-C-r** (read-heavy OLTP; SF=50, 16 terminals, 80% OrderStatus), **TPC-C-w** (write-heavy OLTP; SF=20, 16 terminals, 45% NewOrder and 43% Payment), and **TPC-H** (OLAP; SF=1, 22 serial queries). Unless stated otherwise, each configuration is measured with three repetitions.

Table 4: Performance improvement (%) over default PostgreSQL. Cluster uses the profiling VM environment; server transfers VM-derived knowledge to new hardware. GPTuner uses 100 trials. Gain rows report percentage-point differences.

		Cluster (2 vCPU, 8 GB)			**Server** (192-core, 1 TB)		
System	**Knowledge**	TPC-C-r	TPC-C-w	TPC-H	TPC-C-r	TPC-C-w	TPC-H
GPTuner (LLM & BO)	Original (PG docs)	+9.0	+16.5	−3.2	−54.2	−56.4	−54.7
		68% valid	*70% valid*	*81% valid*			
	+PERFEVOLVE knowledge	+10.4	+19.7	+2.9	−3.3	−10.6	+4.2
		100% valid	*100% valid*	*100% valid*			
	Gain	+1.4	+3.2	+6.1	**+50.9**	**+45.8**	**+58.9**
E2ETune (one-shot)	Original (PG docs)	−1.0	+0.7	+0.0	+6.9	+12.0	+33.6
	+PERFEVOLVE knowledge	+1.0	+0.0	+0.5	+8.4	+12.7	+35.2
	Gain	+2.0	−0.7	+0.5	**+1.5**	**+0.7**	**+1.6**

4.1 END-TO-END PERFORMANCE

Matched environment (VM cluster). Replacing GPTuner's documentation-derived constraints with PERFEVOLVE adds 1.4–6.1 pp across workloads and raises the valid-trial rate from 68%–81% to 100% (Table 4). On TPC-H, performance shifts from 3.2% below default to 2.9% above it. E2ETune shows smaller, mixed changes (−0.7 to +2.0 pp), suggesting that the knowledge is particularly useful when it directly constrains iterative search.

Cross-hardware transfer (server). Original GPTuner performs 54.2%–56.4% below default. Replacing its 57 documentation-derived parameters with 15 sensitivity-ranked parameters and screened ranges adds 45.8–58.9%. TPC-H reaches 4.2% above default, while TPC-C-r and TPC-C-w remain

3.3% and 10.6% below it. Thus, transferred knowledge substantially mitigates degradation but does not eliminate the need for deployment-specific recalibration. E2ETune gains 0.7–1.6% across server workloads, indicating additional benefits beyond its learned tuning knowledge.

4.2 Why Knowledge Form Matters

Table 5: Knowledge variants on server E2ETune, with five runs per condition. Values are performance improvement (%) over default PostgreSQL.

Knowledge	Form	TPC-C-r	TPC-C-w	TPC-H
E2ETune Vanilla	None	+6.9	+12.0	+33.6
Declarative (what to set):				
+ Wrong Numbers	Incorrect values	+0.2	+1.7	−8.3
+ Full Numbers	Correct values	−0.4	−1.9	−2.1
+ HW-Relative	Adapted ratios	+1.0	−1.4	+2.2
Procedural (how to determine):				
+ Structure-Only	Steps, no data	+1.3	−0.1	+5.3
+ Full PerfEvolve	Steps + ref. data	**+8.4**	**+12.7**	**+35.2**

With hardware and workloads fixed, we compare five injected knowledge variants (Table 5). The variants supply fixed values, hardware-relative ratios, procedural steps, or steps with reference data.

Declarative values do not transfer. Full Numbers, although correct in the source environment, yields −2.1% to −0.4% and trails E2ETune Vanilla by 7.3%-35.7%. Hardware-relative scaling does not close the gap, while incorrect values reduce TPC-H by 8.3%. This illustrates the risk of transferring fixed values across environments §2.2.

Procedure and reference data are complementary. Structure-Only outperforms all declarative variants on TPC-C-r and TPC-H, but still trails E2ETune Vanilla. Adding reference data yields the strongest condition: Full PerfEvolve reaches +8.4%, +12.7%, and +35.2%, exceeding Vanilla by 0.7%-1.6%. Thus, transferable procedures and measured priors are complementary.

4.3 Profiling Cost Analysis

Table 6: Offline profiling and online tuning budgets.

Phase (mode)	Runs	%
Offline (per version)		
1. Sensitivity scan (full)	6,297	57
2. Interaction screen (pair)	∼3,500	32
3. Joint optimization (component)	∼1,200	11
Total	**∼11,000**	**100**
Online (per deployment)		
Agent-guided tuning (serial)	30–120	–

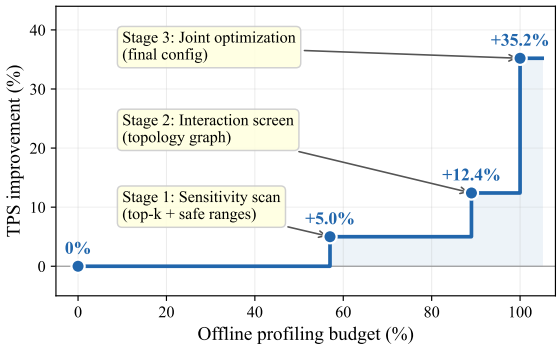


Figure 5: Stage-level cost–performance tradeoff.

Offline profiling requires approximately 11,000 benchmark runs once per system version, while deployment-time tuning uses only 30–120 sequential runs (Table 6). Most offline measurements are parallelizable, so the run count reflects total experimental work rather than deployment latency.

Stage-level return. The sensitivity scan uses 57% of the budget, reduces the search space from 116 to 15 parameters, and yields +5.0%. Interaction screening consumes another 32% and raises the improvement to +12.4%, while joint optimization uses the remaining 11% and reaches +35.2%. The final stage benefits from the component structure discovered earlier.

4.4 Ablation: Contribution of Each Technique

We evaluate five E2ETune conditions on the high-end server, with three repetitions per condition, to isolate the contribution of parameter ranking, interaction knowledge, and safe ranges.

Workload dependence. Ranking performs best on TPC-C-r (+9.5%), while interaction knowledge performs best on TPC-C-w (+13.2%). This pattern is consistent with TPC-C-r depending on a small set of memory/I/O parameters and TPC-C-w stressing coupled buffer, WAL, and checkpoint paths. Full PerfEvolve is best on TPC-H (+35.2%), remains within 1.1 points of the best OLTP result, and provides the highest mean improvement across the three workloads (18.8%). Thus, no isolated component dominates universally, while their combination is the most consistent overall.

Table 7: Knowledge-component ablation on E2ETune (server; three repetitions). Values are improvement (%) over default; **bold** marks the best result per workload.

Condition	TPC-C-r	TPC-C-w	TPC-H
E2ETune Vanilla	+6.9	+12.0	+33.6
+ Ranking only	**+9.5**	+9.0	+30.9
+ Interaction only	+3.3	**+13.2**	+30.1
+ Safe range only	+7.9	+12.6	+32.8
+ Full (PerfEvolve)	+8.4	+12.7	**+35.2**

Joint optimization avoids incompatible local optima. On the VM cluster, independently tuning the `shared_buffers` × `checkpoint_completion_target` pair and combining their local optima decreases TPC-C-r throughput by 13.4%, whereas joint search improves it by 35.2%, a 48.6% gap. This directly shows that detecting interactions is insufficient unless correlated parameters are subsequently optimized together.

5 Related Work

LLM-based system tuning. GPTuner Lao et al. (2024) extracts documentation knowledge for Bayesian optimization, while D-Bot Zhou et al. (2024a) and DB-GPT Zhou et al. (2024b) apply LLMs to diagnosis and root-cause analysis. BYOS Lin et al. (2026), OS-R1 Lin et al. (2025), and Wayfinder Jung et al. (2026) extend LLM-based optimization to operating-system configuration and specialization. These systems generally consume an existing knowledge source; PerfEvolve instead constructs empirically grounded procedural knowledge. Its gains with both GPTuner and E2ETune Huang et al. (2025) indicate that the benefit is not tied to a particular optimizer.

ML-based database tuning. OtterTune Van Aken et al. (2017), CDBTune Zhang et al. (2019), ResTune Zhang et al. (2021), LlamaTune Kanellis et al. (2022), and Hunter Cai et al. (2022) improve parameter selection, search efficiency, or workload transfer through Gaussian processes, reinforcement learning, dimensionality reduction, and transfer learning. PerfEvolve is complementary: it identifies which parameters should be tuned jointly and supplies verifiable procedures that can guide different tuning algorithms (§4).

Configuration interaction analysis. ConfigCrusher Velez et al. (2020), ConEx Krishna et al. (2022), and SPLConqueror Siegmund et al. (2012) characterize parameter interactions for analysis and debugging. PerfEvolve turns such structure into tuning guidance by using factorial screening and replicated ANOVA to construct joint-search components and executable skills.

Operational knowledge management. SRE runbooks and playbooks encode operational procedures Beyer et al. (2016), and recent work explores their execution with LLMs Smith (2026). They motivate procedural representations but target incident response; PerfEvolve focuses on performance tuning and empirically calibrates its decision criteria.

6 Conclusion

System tuning guides often record recommendations while leaving the decision process implicit. We advocate documenting *how to determine* a configuration through reproducible measurements and decision rules. PerfEvolve operationalizes this approach by compiling empirical profiles into executable skills for LLM-based tuning agents. Our results show that combining procedures with reference data can improve tuning reliability and cross-hardware adaptation, while deployment-specific validation remains necessary.

AI USE STATEMENT

We used LLMs to assist with language editing, manuscript organization, and critical feedback on writing. The authors take full responsibility for the accuracy, originality, and integrity of the final manuscript and accompanying artifacts.

ETHICS STATEMENT

This work studies database configuration tuning using standard benchmark workloads and generated data, without human participants or personal data. Automated configuration changes can degrade performance or disrupt service. Our procedures include empirical range screening and deployment-time validation, but these checks do not guarantee safety outside the evaluated settings. Production use should therefore include controlled testing, monitoring, and rollback mechanisms.

REPRODUCIBILITY STATEMENT

Section 3 describes the profiling procedures, interaction tests, thresholds, and procedural knowledge representation. Section 4 reports the hardware, workloads, baselines, trial budgets, and repetition counts. Section 4.3 details the profiling cost, while Sections 4.2 and 4.4 describe the knowledge comparisons and component ablations. Code and artifacts are available at `https://anonymous.4open.science/r/PerfEvolve-DF10`.

REFERENCES

FreeBSD Documentation, 2026. URL `https://docs.freebsd.org/`.

MySQL Reference Manual: Chapter 10 Optimization, 2026. URL `https://dev.mysql.com/doc/refman/8.4/en/optimization.html`.

Redis configuration parameters, 2026a. URL `https://redis.io/docs/latest/develop/ai/search-and-query/administration/configuration/`.

Optimizing Redis, 2026b. URL `https://redis.io/docs/latest/operate/oss_and_stack/management/optimization/`.

Windows technical documentation for developers and IT pros, 2026. URL `https://learn.microsoft.com//windows/`.

Apache Software Foundation. *Apache Kafka 3.6 Documentation*, 2026. URL `https://kafka.apache.org/documentation/#3.6`.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300, 1995.

Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016. URL `http://landing.google.com/sre/book.html`.

Baoqing Cai, Yu Liu, Ce Zhang, Guangyu Zhang, Ke Zhou, Li Liu, Chunhua Li, Bin Cheng, Jie Yang, and Jiashu Xing. Hunter: An online cloud database hybrid tuning system for personalized requirements. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pp. 646–659, 2022. doi: 10.1145/3514221.3517882.

Sibei Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, and Xiaoyong Du. Automatic database configuration debugging using retrieval-augmented language models. *Proc. ACM Manag. Data*, 3(1), 2025. doi: 10.1145/3709663.

Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988.

Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. OLTP-Bench: An extensible testbed for benchmarking relational databases. In *Proc. VLDB Endow.*, volume 7, pp. 277–288, 2013. doi: 10.14778/2732240.2732246.

Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning database configuration parameters with ituned. In *Proc. VLDB Endow*, volume 2, pp. 1246–1257, August 2009. doi: 10.14778/1687627.1687767.

Victor Giannakouris and Immanuel Trummer. λ-tune: Harnessing large language models for automated database system tuning. *Proc. ACM Manag. Data*, 3(1), 2025. doi: 10.1145/3709652.

Xinmei Huang, Haoyang Li, Jing Zhang, Xinxin Zhao, Zhiming Yao, Yiyan Li, Tieying Zhang, Jianjun Chen, Hong Chen, and Cuiping Li. E2ETune: End-to-end knob tuning via fine-tuned generative language model. In *Proc. VLDB Endow.*, 2025.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, pp. 507–523, 2011. doi: 10.1007/978-3-642-25566-3_40.

Alexander Jung, Cezar Crăciunoiu, Nikolaos Karaolidis, Hugo Lefeuvre, Daniel Oñoro Rubio, Felipe Huici, Charalampos Rotsos, and Pierre Olivier. Wayfinder: Automated operating system specialization. In *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, pp. 710–727. ACM, April 2026. doi: 10.1145/3767295.3803589. URL http://dx.doi.org/10.1145/3767295.3803589.

Konstantinos Kanellis, Ramnatthan Alagappan, and Shivaram Venkataraman. Too many knobs to tune? towards faster database tuning by pre-selecting important knobs. In *Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage '20, 2020.

Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. LlamaTune: Sample-efficient DBMS configuration tuning. In *Proc. VLDB Endow.*, volume 15, pp. 2953–2965, 2022. doi: 10.14778/3551793.3551844.

Rahul Krishna, Chong Tang, Kevin Sullivan, and Baishakhi Ray. ConEx: Efficient exploration of big-data system configurations for better performance. In *IEEE Transactions on Software Engineering*, volume 48, pp. 893–909, 2022. doi: 10.1109/TSE.2020.3007560.

Brian Kroth, Sergiy Matusevych, and Yiwen Zhu. Autotuning systems: Techniques, challenges, and opportunities. In *Companion of the 2025 International Conference on Management of Data*, SIGMOD/PODS '25, pp. 821–828, 2025. doi: 10.1145/3722212.3725638.

Mayuresh Kunjir and Shivnath Babu. Black or white? how to develop an autotuner for memory-based analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pp. 1667–1683, 2020. doi: 10.1145/3318464.3380591.

Daniël Lakens. Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and anovas. *Frontiers in Psychology*, 4:863, 2013. doi: 10.3389/fpsyg.2013.00863.

Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. GPTuner: A manual-reading database tuning system via GPT-guided Bayesian optimization. In *Proc. VLDB Endow.*, volume 17, pp. 1939–1952, 2024.

Yiyan Li, Haoyang Li, Jing Zhang, Renata Borovica-Gajic, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Cuiping Li, and Hong Chen. Agenttune: An agent-based large language model framework for database knob tuning. *Proc. ACM Manag. Data*, 3(6), 2025. doi: 10.1145/3769758.

Hongyu Lin, Yuchen Li, Haoran Luo, Kaichun Yao, Libo Zhang, Mingjie Xing, and Yanjun Wu. Os-r1: Agentic operating system kernel tuning with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.12551.

Hongyu Lin, Yuchen Li, Haoran Luo, Kaichun Yao, Libo Zhang, Zhenghong Lin, Mingjie Xing, Yanjun Wu, and Carl Yang. Byos: Knowledge-driven large language models bring your own operating system more excellent, 2026. URL https://arxiv.org/abs/2503.09663.

Linux Kernel Developers. *The kernel's command-line parameters*, 2026. URL https://docs.kernel.org/admin-guide/kernel-parameters.html.

Meta Platforms, Inc. *RocksDB Documentation*, 2026a. URL https://github.com/facebook/rocksdb/wiki.

Meta Platforms, Inc. *RocksDB Tuning Guide*, 2026b. URL https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide.

Oracle Corporation. *MySQL 8.0 Reference Manual*, 2024. URL https://dev.mysql.com/doc/refman/8.0/en/.

Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. Self-driving database management systems. In *Conference on Innovative Data Systems Research*, 2017.

Andrew Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. Make your database system dream of electric sheep: towards self-driving operation. *Proc. VLDB Endow.*, 14(12):3211–3221, 2021. doi: 10.14778/3476311.3476411.

PGTune. PGTune: Postgresql configuration wizard. https://pgtune.leopard.in.ua/, 2024.

PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*, 2023. URL https://www.postgresql.org/docs/16/.

Postgresqlco.nf. Postgresqlco.nf. https://postgresqlco.nf/, 2025.

Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. The case for automatic database administration using deep reinforcement learning, 2018. URL https://arxiv.org/abs/1801.05643.

Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pp. 167–177. IEEE Press, 2012.

Bradley D Smith. An agent-graph workflow built with graph-flow in rust that parses markdown runbooks and executes them step-by-step, using a local llm (ollama) to understand intent and verify outcomes. https://github.com/bradleyd/runbook-executor, 2026.

Wenwen Sun, Zhicheng Pan, Zirui Hu, Yu Liu, Chengcheng Yang, Rong Zhang, and Xuan Zhou. Rabbit: Retrieval-augmented generation enables better automatic database knob tuning. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 3807–3820, 2025. doi: 10.1109/ICDE65448.2025.00284.

The PostgreSQL Global Development Group. *PostgreSQL: Chapter 14. Performance Tips*, 2026. URL https://www.postgresql.org/docs/current/performance-tips.html.

Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 International Conference on Management of Data*, SIGMOD '17, pp. 1009–1024, 2017. doi: 10.1145/3035918.3064029.

Miguel Velez, Pooyan Jamshidi, Florian Sattler, Norbert Siegmund, Sven Apel, and Christian Kästner. ConfigCrusher: Towards white-box performance analysis for configurable systems. In *Automated Software Engg.*, volume 27, pp. 265–300. Kluwer Academic Publishers, 2020. doi: 10.1007/s10515-020-00273-8.

Francesco Ventura, Zoi Kaoudi, Jorge Arnulfo Quiané-Ruiz, and Volker Markl. Expand your training limits! generating training data for ml-based data management. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pp. 1865–1878, 2021. doi: 10.1145/3448016.3457286.

Xinyue Yang, Chen Zheng, Yaoyang Hou, Renhao Zhang, Yinyan Zhang, Yanjun Wu, and Heng Zhang. L2t-tune:llm-guided hybrid database tuning with lhs and td3, 2025. URL https://arxiv.org/abs/2511.01602.

Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pp. 415–432, 2019. doi: 10.1145/3299869.3300085.

Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. ResTune: Resource oriented tuning boosted by meta-learning for cloud databases. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pp. 2102–2114, 2021. doi: 10.1145/3448016.3457291.

Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. D-Bot: Database diagnosis system using large language models. In *Proc. VLDB Endow.*, volume 17, 2024a. doi: 10.14778/3675034.3675043.

Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. DB-GPT: Large language model meets database. In *Data Science and Engineering*, volume 9, pp. 102—111, 2024b. doi: 10.1007/s41019-023-00235-6.

Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pp. 338–350, 2017. doi: 10.1145/3127479.3128605.

APPENDIX

A EXPERIMENTAL CONFIGURATION AND EVALUATION SCOPE

A.1 TESTBEDS AND WORKLOADS

Table 8 summarizes the two environments used in Section 4. The matched setting uses the same VM specification for offline profiling and deployment evaluation, with SSD storage. The transfer setting applies VM-derived knowledge to a substantially larger server. The H100 GPUs support E2ETune inference; PostgreSQL execution uses the server's CPU, memory, and storage resources.

Table 8: Reported testbed configuration.

	Matched environment	**Transfer environment**
Database	PostgreSQL 16	PostgreSQL 16
Compute	150 VMs, 2 vCPUs each	192-core Xeon 8575C server
Memory	8 GB per VM	1 TB
Role	Offline profiling and matched evaluation	Cross-hardware evaluation

We use BenchBase with two modified TPC-C transaction mixes and one TPC-H workload (Table 9). These experiments evaluate tuning under the stated configurations; they are not claims of audited TPC benchmark compliance. Unless an experiment specifies otherwise, each configuration is measured three times.

Table 9: Workload settings reported in the main paper.

Workload	**Scale factor**	**Concurrency**	**Workload composition**
TPC-C-r	50	16 terminals	80% OrderStatus; read-heavy mix
TPC-C-w	20	16 terminals	45% NewOrder and 43% Payment; write-heavy mix
TPC-H	1	Serial queries	All 22 benchmark queries

A.2 INTEGRATION MODES AND REPETITION COUNTS

GPTuner uses documentation-derived constraints to guide SMAC. The reported integration replaces these priors with selected parameters, screened ranges, and interaction annotations while retaining the optimizer. The parameter count changes from 57 to 15, and the main comparison uses 100 trials per condition. Thus, this comparison measures the benefit of the supplied knowledge and induced search space with a fixed optimizer and online trial budget; it does not equalize offline profiling effort.

E2ETune generates a 44-parameter configuration in one inference. Its integration changes the supplied knowledge while retaining the model. Providing procedural descriptions to a one-shot model does not itself demonstrate benchmark-driven traversal of the skill DAG. Full procedural execution and knowledge injection are therefore separate deployment modes.

The knowledge-variant comparison reports five runs per condition, whereas the component ablation reports three repetitions. These counts belong to different experiments and should not be pooled. Configuration-generation variability and repeated measurement of a fixed configuration are distinct sources of variation.

B SENSITIVITY PROFILES AND THEIR INTERPRETATION

Let $\mathcal{V}_i$ contain the L_i tested values of parameter x_i, and let $\mathbf{x}^{(0)}$ be the default configuration. For workload w, $y_{i,v,w,t}$ denotes throughput on repetition t with $x_i = v$ and all other parameters at their defaults. The mean response is

$$\bar{y}_{i,v,w} = \frac{1}{r} \sum_{t=1}^{r} y_{i,v,w,t}. \tag{3}$$

Writing $\bar{y}_{0,w}$ for default throughput, the sensitivity score in Section 3.1 is

$$S_{i,w} = \frac{\max_{v \in \mathcal{V}_i} \bar{y}_{i,v,w} - \min_{v \in \mathcal{V}_i} \bar{y}_{i,v,w}}{\bar{y}_{0,w}}, \qquad S_i = \max_w S_{i,w}. \tag{4}$$

Here $S_{i,w}$ is the quantity denoted by $\mathrm{CV}_{i,w}$ in the main text. It is a normalized response range, not the conventional coefficient of variation (standard deviation divided by the mean). The calculation requires positive default throughput and numerical measurements for the evaluated configurations.

The maximum across workloads retains a parameter that matters for at least one evaluated workload. Scores above $\tau_s = 0.05$ are ranked to select the retained parameters. The reported PostgreSQL scan retains 15 of 116 parameters. This is evidence about the sampled values, workloads, and default operating point, rather than a guarantee that the remaining parameters are irrelevant under every configuration.

Screened ranges. The reported screening excludes observed crashes and throughput losses greater than 50% relative to default. For a successfully measured setting, the latter corresponds to $\bar{y}_{i,v,w} < 0.5\bar{y}_{0,w}$. These are empirical exclusions: successful evaluation of sampled values does not establish the safety of every intermediate value, every joint configuration, or a new deployment. Failure records must be distinguished from numerical throughput observations.

Local screening limitation. Because the scan changes one parameter at a time, it can miss a parameter whose effect appears only when another parameter changes. Subsequent interaction discovery is restricted to the retained set and cannot recover every interaction involving a discarded parameter. The procedure therefore trades search coverage for a smaller profiling and tuning space.

C Factorial Interaction Analysis

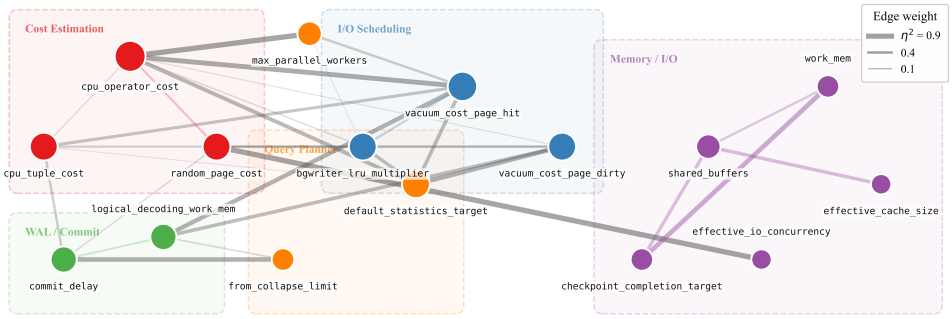


Figure 6: PostgreSQL interaction graph for `TPC-C-r`. Edges denote confirmed interactions and their widths encode η^2; colors identify connected components (left). Each component induces a joint-search subproblem, while isolated parameters are searched independently (right).

C.1 Coarse Interaction Contrasts

For a parameter pair, the two extreme tested values define the low and high levels of a 2×2 design. Let y_{ab} be the single throughput observation at levels $a, b \in \{-, +\}$. Define

$$D_{ij} = y_{++} - y_{+-} - y_{-+} + y_{--}, \qquad \bar{y} = \frac{y_{++} + y_{+-} + y_{-+} + y_{--}}{4}. \tag{5}$$

The coarse score is $I_{ij} = |D_{ij}|/|\bar{y}|$, provided $\bar{y} \neq 0$. For an additive response $y_{ab} = \mu + \alpha_a + \beta_b$, the contrast is zero. A nonzero contrast can reflect an interaction or measurement noise. With one observation per cell, the saturated two-factor model has no residual degrees of freedom for an interaction significance test.

The reported candidate threshold is $I_{ij} > 0.15$, equivalent to a score above 15% when displayed in percent units. Stage A prioritizes pairs for replicated testing; a low score is not proof of independence.

C.2 Replicated two-factor ANOVA

Stage B uses $a = b = 4$ levels and $r = 3$ repetitions per cell. For complete, balanced observations, the standard model underlying this analysis is

$$y_{uvt} = \mu + \alpha_u + \beta_v + (\alpha\beta)_{uv} + \epsilon_{uvt}, \quad u = 1, \ldots, a, \quad v = 1, \ldots, b, \quad t = 1, \ldots, r. \tag{6}$$

Let $\bar{y}_{uv\cdot}$, $\bar{y}_{u\cdot\cdot}$, $\bar{y}_{\cdot v\cdot}$, and $\bar{y}_{\cdots}$ denote the cell, row, column, and grand means. The relevant sums of squares are

$$\mathrm{SS}_{ij} = r \sum_{u=1}^{a} \sum_{v=1}^{b} (\bar{y}_{uv\cdot} - \bar{y}_{u\cdot\cdot} - \bar{y}_{\cdot v\cdot} + \bar{y}_{\cdots})^2, \tag{7}$$

$$\mathrm{SS}_E = \sum_{u=1}^{a} \sum_{v=1}^{b} \sum_{t=1}^{r} (y_{uvt} - \bar{y}_{uv\cdot})^2, \tag{8}$$

$$\mathrm{SS}_T = \sum_{u=1}^{a} \sum_{v=1}^{b} \sum_{t=1}^{r} (y_{uvt} - \bar{y}_{\cdots})^2. \tag{9}$$

The interaction effect size and test statistic are

$$\eta^2_{ij} = \frac{\mathrm{SS}_{ij}}{\mathrm{SS}_T}, \qquad F_{ij} = \frac{\mathrm{SS}_{ij}/[(a-1)(b-1)]}{\mathrm{SS}_E/[ab(r-1)]}. \tag{10}$$

For the reported design, the numerator and denominator degrees of freedom are 9 and 32. This η^2 uses total variation as its denominator and differs from partial η^2. Its interpretation is restricted to the tested levels and workload. The usual F reference distribution assumes independent, approximately normal errors with common variance; incomplete cells require different handling.

C.3 Multiple testing and graph construction

For a family of m confirmation tests, order their raw p-values as $p_{(1)} \leq \cdots \leq p_{(m)}$. The Benjamini–Hochberg adjusted values are

$$q_{(\ell)} = \min \left\{ 1, \min_{t \geq \ell} \frac{m}{t} p_{(t)} \right\}. \tag{11}$$

After mapping these values back to the parameter pairs, the reported edge rule retains pairs with $q_{ij} < 0.05$ and $\eta^2_{ij} > 0.15$. Statistical significance and effect magnitude are separate requirements. Multiplicity correction addresses the tested family; it does not recover interactions discarded by coarse screening.

The resulting weighted, undirected parameter graph is distinct from the directed skill DAG. Its connected components identify joint-search groups; the skill DAG orders the experiments and checks used to tune those groups. Disconnected components imply no retained pairwise edge between them, not proven statistical independence or the absence of higher-order effects.

D Procedural Knowledge: Structure and Example

Table 10 expands the skill contents described in Section 3.3. It is a conceptual specification, not a literal serialization schema or a claim about additional implemented tools.

Illustrative component skill. Consider `shared_buffers` and `checkpoint_completion_target` from the case study. This example illustrates the representation; it is not an exported prompt or an additional experimental result.

1. **Check context.** Identify the target workload, available resources, current settings, and relevant offline profile.
2. **Establish a reference.** Measure the default or explicitly designated comparison configuration on the target deployment.

Table 10: Conceptual contents of a procedural skill.

Field	Purpose
Preconditions	Conditions under which the skill's actions and reference data apply.
Measurements	Observations needed to evaluate candidates and check assumptions.
Actions	Configuration changes and experiments prescribed by the procedure.
Decision rules	Criteria for choosing the next action or accepting a candidate.
Postconditions	Checks on the resulting configuration and its measured behavior.
Reference data	Offline observations that inform, but do not replace, local measurements.

3. **Evaluate jointly.** Run the prescribed candidate combinations within the component's screened search space and record their outcomes.

4. **Apply the decision rule.** Select a measured candidate using the skill's objective and acceptance criteria. A failed check must be handled explicitly rather than interpreted as a valid performance result.

5. **Verify scope.** Check postconditions and evaluate the selected configuration on any additional workloads required by the procedure.

Candidate values, budgets, and acceptance thresholds are deployment inputs; this illustration does not assign unreported numerical values.

Why the cross-workload check matters. In the reported VM experiment, joint tuning of this pair improves TPC-C-r throughput by 35.2%, but applying the same configuration to TPC-C-w and TPC-H produces changes of -4.3% and -18.6%, respectively. The procedure must therefore preserve the workload context of its evidence. A setting selected for one workload is not automatically a validated recommendation for another.

Knowledge injection. GPTuner consumes the exported parameters, ranges, and interaction annotations through its existing optimization interface. E2ETune consumes injected knowledge through its input context. These integrations reuse information derived from the procedural representation, while full execution additionally requires an agent to perform the prescribed measurements and checks.

E KNOWLEDGE VARIANTS AND RESULT ACCOUNTING

E.1 INTERPRETATION OF THE KNOWLEDGE COMPARISONS

The knowledge-form experiment compares Vanilla with five injected variants: incorrect values (Wrong Numbers), values valid in the source environment (Full Numbers), hardware-relative ratios (HW-Relative), procedural steps without reference measurements (Structure-Only), and steps with reference data (Full PERFEVOLVE). These variants change both the representation and, in some cases, the information supplied. Consequently, the experiment evaluates those knowledge packages, rather than isolating a formatting effect under identical information content.

Structure-Only outperforms the declarative variants on TPC-C-r and TPC-H but remains below Vanilla. Full PERFEVOLVE exceeds Vanilla on all three server workloads. Together, these observations support the combination of procedures and empirical reference data within the evaluated setting; they do not establish that procedural instructions alone always improve a model's tuning decisions.

The component ablation adds ranking, interaction knowledge, or safe ranges individually and compares them with the full combination. These are single-component additions, not leave-one-component-out experiments. They reveal workload-dependent usefulness but do not uniquely identify the marginal contribution of each component within the full system.

E.2 Percentages and percentage-point differences

For throughput T relative to default throughput T_0, improvement is $I = 100(T/T_0 - 1)$ percent. If two results are reported as improvements I_A and I_B over the same default, their difference

$$\Delta_{\text{pp}} = I_A - I_B \tag{12}$$

is measured in percentage points. It is not the relative improvement of configuration A over configuration B. For a higher-is-better metric normalized as above, that relative improvement would be

$$\Delta_{\text{rel}} = 100 \left(\frac{1 + I_A/100}{1 + I_B/100} - 1 \right) \%. \tag{13}$$

Table 11 shows arithmetic differences already implicit in the reported results; it introduces no new measurements.

Table 11: Examples of percentage-point accounting from the main results.

Comparison	**Before (%)**	**After (%)**	**Difference (pp)**
GPTuner, cluster TPC-H	-3.2	$+2.9$	$+6.1$
GPTuner, server TPC-C-w	-56.4	-10.6	$+45.8$
E2ETune, server TPC-H	$+33.6$	$+35.2$	$+1.6$
Independent vs. joint, VM TPC-C-r	-13.4	$+35.2$	$+48.6$

The server E2ETune TPC-H result and the VM pairwise TPC-C-r result both equal 35.2%, but refer to different experiments. Likewise, the reported 18.8% ablation summary is the arithmetic mean $(8.4 + 12.7 + 35.2)/3$, rounded to one decimal place; it is not a pooled throughput gain or a geometric-mean speedup.

F Profiling and Search Complexity

With L_i sensitivity values, r repetitions, and W workloads, the nominal sensitivity-scan cost is

$$N_{\text{sens}} = rW \sum_{i=1}^{n} L_i. \tag{14}$$

For $n = 116$, $r = W = 3$, the reported 6,297 runs correspond to $\sum_i L_i = 699$, or about 6.03 levels per parameter.

For k retained parameters and one observation per Stage A cell, a full pairwise screen costs

$$N_A = 4W \binom{k}{2}. \tag{15}$$

If m_w pairs advance on workload w, a complete 4×4 confirmation design with three repetitions costs

$$N_B = 48 \sum_{w=1}^{W} m_w. \tag{16}$$

The illustrative 11-parameter TPC-C-r screen in Section 2.2 contains 55 pairs, of which 33 are coarse candidates. Its screening cost is $55 \times 4 = 220$ observations; confirming all 33 candidates would require $33 \times 48 = 1,584$ additional observations for that workload. The latter is a design-based calculation, not an additional reported execution count. This 11-parameter illustration is distinct from the 15-parameter retained set, which contains 105 possible pairs.

Component search. Suppose the retained graph has component sizes $d_1, \ldots, d_M$, including isolated vertices, with $\sum_j d_j = k$. An exhaustive q-level search performed separately within each component visits

$$N_{\text{component}} = \sum_{j=1}^{M} q^{d_j} \tag{17}$$

candidate settings per workload and per single measurement pass, compared with q^k for a full Cartesian search. Repetitions, repeated optimization passes, and final validation add to this cost. Thus, $4^3 = 64$ describes one three-parameter component, not the entire tuning budget. This decomposition is an approximation based on detected interactions and does not guarantee the optimum of the full space.

Amortization. The reported offline phases total approximately 11,000 runs, while full agent-guided deployment uses 30–120 sequential runs. If one offline profile remains applicable to D deployments without reconstruction, the average experimental cost is

$$\bar{N}(D) = \frac{N_{\text{offline}} + \sum_{d=1}^{D} N_{\text{online},d}}{D}. \tag{18}$$

This is a conditional accounting identity, not a measured break-even result. Additional recalibration costs must be included when they occur. Parallelism reduces elapsed time but does not remove experimental work. The 30–120-run full-execution budget is also distinct from E2ETune's single inference and GPTuner's trial budget in the integration experiments.

G Scope and Limitations

Coverage. The quantitative evaluation concerns PostgreSQL 16, three benchmark workloads, and two hardware environments. The documentation examples from Linux, MySQL, and Kafka motivate the problem but do not establish measured tuning gains on those systems. The matched testbed is not an unseen-hardware test, and the large server does not by itself establish effectiveness at larger database scale factors.

Statistical and search limitations. Finite sampling, one-factor screening, and pairwise tests can miss narrow response regions and higher-order interactions. Effect sizes depend on the selected factor levels. A 100% valid-trial rate in the observed matched-environment runs is an empirical result, not a guarantee for future workloads or deployments. Small performance differences require run-level variability estimates to assess their reliability.

Transfer and deployment. Transferred GPTuner knowledge improves over the original tuner but remains below default on both server TPC-C workloads. Procedures and reference data therefore require deployment-specific checks. The current experiments do not establish a universal recalibration cost or that one profile remains sufficient for every deployment of a software version.

Attribution. The integrations evaluate a package of empirical measurements, parameter selection, search constraints, and procedural descriptions. Their gains should not all be attributed to representation alone. Separating these effects further would require matched-information comparisons and additional controlled experiments, which are outside the reported study.