

Query Cost Model Calibration in Confidential Virtual Machines

Qihan Zhang
University of Southern California
Los Angeles, USA
qihanzha@usc.edu

Mengyuan Li
University of Southern California
Los Angeles, USA
mli49061@usc.edu

Ibrahim Sabek
University of Southern California
Los Angeles, USA
sabek@usc.edu

Abstract

With the growing adoption of Confidential Computing, running databases in confidential virtual machines (CVMs) such as AMD SEV-SNP has become an attractive way to protect sensitive cloud data with minimal changes to legacy DBMSs. However, analytical queries in such CVMs often suffer substantial overhead, and prior database work has largely stopped at benchmarking these slow-downs rather than optimizing them. We show that this problem stems from a hardware–software mismatch: query optimizers still rely on KVM-oriented (non-encrypted VM) cost assumptions that no longer hold in CVMs. To address this, we propose a lightweight CVM-aware cost calibration. It models two dominant sources of optimizer-facing overhead: data-movement and RMP-related translation using simple physical proxies already available to the optimizer. Experiments show that the calibration significantly narrows the KVM/CVM performance gap, recovering up to 48% performance and even outperforming the KVM baseline on some workloads.

ACM Reference Format:

Qihan Zhang, Mengyuan Li, and Ibrahim Sabek. 2026. Query Cost Model Calibration in Confidential Virtual Machines. In *Workshop on Secure and Private Data Management (SeQureDB '26)*, May 31–June 05, 2026, Bengaluru, India. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3807894.3810270>

1 Introduction

As cloud infrastructure increasingly hosts critical database workloads on third-party platforms [4, 12, 42], protecting sensitive data without sacrificing performance has become a central challenge [10, 15, 36]. Confidential computing addresses this by protecting data in use within Trusted Execution Environments (TEEs). TEEs broadly fall into two classes: process-based designs such as Intel SGX [11], which offer strong isolation but often require intrusive application changes, and VM-based designs such as AMD SEV-SNP [41], which protect entire virtual machines with better compatibility but a larger trusted software stack. We focus on the latter, namely Confidential Virtual Machines (CVMs), such as AMD SEV-SNP, because they let legacy DBMSs like PostgreSQL [38] run with no source-code change, making them attractive for practical deployment [3, 5, 21, 32, 35]. Throughout the paper, we use *CVM* to denote a hardware-protected confidential VM, and *KVM* to denote the conventional unencrypted VM baseline.

However, this convenience comes with non-trivial overheads in memory protection, address translation, and I/O [2, 32]. Prior

studies on DBMSs in CVMs mainly focus on benchmarking these penalties [6, 25, 26, 28, 39]. While they show that CVMs can substantially slow down database workloads, they do not examine how such overhead distorts core DBMS components, especially the query optimizer. Since optimizers choose among scan paths, join methods, and execution structures by comparing estimated costs [16, 17, 40], a CVM-unaware cost model can misrank operators and produce poor plans. This creates a hardware–software mismatch: cost estimations for KVMs no longer hold in CVMs.

Focusing on AMD SEV-SNP, we attribute this mismatch to two architectural properties of CVM (empirically evaluated in Section 2). First, data movement in a CVM more often involves encryption-related processing and extra copying across the storage–memory hierarchy. This gives rise to *data movement overhead*, including bounce-buffer copying on I/O paths and the higher cost of transferring large volumes of encrypted data under memory- or bandwidth-intensive execution [33, 34]. Second, SEV-SNP introduces the Reverse Map Table (RMP), a hardware metadata structure that validates page ownership and access permissions during address translation (i.e., mapping virtual addresses to physical pages). This gives rise to *RMP overhead*, which raises the cost of memory-access paths triggering page-table walks and metadata checks, making random-access operators such as index scans and probe-heavy nested-loops more expensive [1, 41, 44, 45]. These penalties are highly non-uniform across operators, so plans that are reasonable in KVM can become poor choices in SEV-SNP. To address this problem, we propose a lightweight CVM-aware calibration centered on the query optimizer’s *cost model*. Rather than redesigning the execution engine, it adjusts operator costing so that the DBMS better reflects the true relative prices of scan paths and join strategies in SEV-SNP. The key idea is to use simple physical proxies already exposed to the optimizer, such as pages touched, tuple volume, and memory usage, to inject targeted penalties for RMP-sensitive random access and data movement. This preserves the optimizer’s architecture while making plan decisions more faithful to CVMs. This calibration introduces essentially negligible overhead, as it only lightly modifies the optimizer’s cost estimation logic without adding new execution-time mechanisms.

We implement the calibration in PostgreSQL and evaluate it on standard OLAP benchmarks (JOB [23], CEB [37], Stack [30], and TPC-DS [8]), saving at most 48% execution time in CVMs. In summary, our contributions are:

- (1) We show that RMP and data movement overheads explain the main optimizer-facing performance distortions relevant to query planning. Considering them in the cost model can improve DBMS performance in CVMs.
- (2) We propose a CVM-calibrated cost model, which is a practical CVM-aware cost-model calibration extension that significantly



This work is licensed under a Creative Commons Attribution 4.0 International License. *SeQureDB '26, Bengaluru, India*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2219-6/2026/05
<https://doi.org/10.1145/3807894.3810270>

narrows the KVM/CVM performance gap and can even outperform the KVM baseline.

- (3) We implement the CVM-aware calibration inside PostgreSQL for CVMs, recovering up to 48% performance and even outperforming the non-encrypted baseline on some workloads.

2 Motivation

SEV-SNP introduces at least two optimizer-relevant penalties, both rooted in its security mechanisms. The first is *data movement cost*. In a CVM, transfers along the storage–memory path are no longer plain copies: they typically involve encrypted guest memory, additional I/O copying, and synchronization through intermediate software or hardware layers. A key mechanism is the *bounce buffer*, a shared staging area used when data cannot be moved directly between devices and protected guest memory [33]. Since different query plans stress this path differently, the resulting overhead is strongly plan-dependent. The second is *RMP-check cost*. SEV-SNP protects guest memory with additional metadata and validation, commonly tied to the Reverse Map Table (RMP) [41, 45]. On translation-related events such as TLB or cache misses, the processor may perform page-table walks and protected-memory checks before completing the access. As a result, SEV-SNP can raise address-translation and memory-access latency even when the query plan is identical between KVM and CVM.

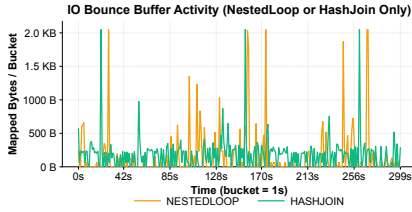


Figure 1: Evidence of *data movement cost* in SEV-SNP. Hash join induces denser bounce-buffer activity, while nested-loop join induces less frequent but burstier activity.

Experimental Evidence. Figure 1 and Table 1 show that both penalties are observable. To expose the *data movement cost*, we repeatedly run JOB [23] in SEV-SNP under two settings, *hash-join-only* and *nested-loop-only* (disable other join strategies), with all other settings unchanged. For each setting, we monitor bounce-buffer activity for 300 seconds, partition the trace into 1-second buckets, and record transferred bytes (*mapped bytes*) per bucket. The figure reveals clear operator-dependent movement patterns: hash join yields denser, more continuous activity, consistent with sustained movement during scans and hash build/probe phases, whereas nested loop is less frequent but more bursty, matching its operator behavior. This shows that SEV-SNP overhead is not a fixed slowdown but depends on how a plan moves data, motivating a *data movement cost* in the operator cost model. To expose the *RMP-check cost*, we run the *same SQL (JOB 17a)* under the *same query plan* in KVM and SEV-SNP and collect low-level counters. The key signal is CPI (Cycles Per Instruction) inflation in SEV-SNP: instructions grow by only 1.23 \times , while cycles grow by 2.25 \times , raising CPI from 1.28 to 2.33. This indicates that SEV-SNP mainly increases the *cost per instruction* rather than the amount of work performed.

Table 1: Evidence of *RMP-check cost* in SEV-SNP. For the same SQL and plan, SEV-SNP shows clear CPI inflation relative to KVM.

Metric	KVM	SEV-SNP	\times (SNP/KVM)
Exec time (s)	14.85	18.26	1.23
Cycles ($\times 10^{11}$)	1.22	2.75	2.25
Instructions ($\times 10^{11}$)	0.96	1.18	1.23
CPI (cycles/inst)	1.28	2.33	1.82
dTLB misses/cycle ($\times 10^{-4}$)	1.51	1.93	1.28
iTLB misses/cycle ($\times 10^{-3}$)	1.11	3.30	2.98
Cache misses/cycle ($\times 10^{-3}$)	2.65	2.09	0.79

The pattern is consistent with translation-related slowdown: TLB misses trigger page-table walks, and protected-page accesses may incur extra RMP validation. Miss metrics do not indicate a miss-rate explosion. dTLB misses per cycle rise only modestly, as expected, since the same plan should preserve broadly similar data-access locality across KVM and CVM. iTLB misses per cycle increase more noticeably, suggesting higher instruction-side translation cost under RMP checks. Cache misses per cycle slightly decrease, which is also compatible with SEV-SNP-induced stall amplification: when total cycles grow substantially, longer stall time per miss can dilute miss rates normalized by cycle. Overall, the evidence is consistent with *latency amplification on existing translation and memory-access paths*, confirming the existence of *RMP-check cost*.

3 Overview of CVM-calibrated Cost Model

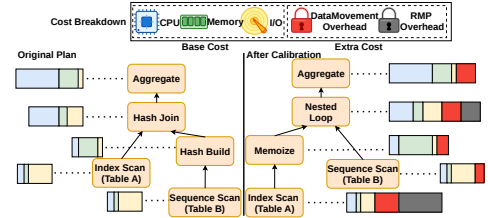


Figure 2: Our light-weight CVM-aware calibration adds two extra overheads for operator cost modeling in CVM.

Cost modeling is the crucial component behind cost-based query optimization: it ranks alternative operators and join orders by predicting execution cost. In CVMs such as SEV-SNP, encryption and integrity mechanisms reshape the relative prices of CPU, memory, and I/O, so conventional cost models can become systematically miscalibrated and prefer CVM-fragile plans. We therefore augment the DBMS’s base cost formulas with a small number of CVM-aware terms. As illustrated in Figure 2, traditional DBMS cost models decompose operator cost into CPU, memory, and I/O components [29], but do not account for the CVM-specific overheads discussed earlier. Our motivation results in Section 2 show that these overheads are not uniform across operators. On one hand, different join strategies induce very different bounce-buffer activity patterns, indicating operator-dependent data movement overhead. On the other hand, even under the same SQL and unchanged plan, SEV-SNP still exhibits clear CPI inflation, pointing to additional translation and protected-memory-access cost. These observations motivate our

Table 2: a light-weight CVM-aware calibration for main operators.

Operator	Formula (Base cost with CVM overhead)	Operator	Formula (Base cost with CVM overhead)
Sequential scan (SS)	$C_{SS}(R) = C_{SEQ}^{BASE}(R) + \text{DATAMOVECOST}(ws^{SS}(R), x(R))$	Bitmap scan (BS)	$C_{BS}(R) = C_{BS}^{BASE}(R) + \text{DATAMOVECOST}(ws^{BS}(R), x(R))$
Index (only) scan [I(O)S]	$C_{I(O)S}(R, I) = C_{I(O)S}^{BASE}(R, I) + \text{RMPTCOST}(rows_{OUT}(R) \cdot H(I), x(R, I)) + \text{DATAMOVECOST}(ws^{I(O)S}(R, I), x(R, I))$	Nested-loop join (NL)	$C_{NL}(R, S) = C_{NL}^{BASE}(R, S) + \text{DATAMOVECOST}(ws^{NL}(R, S), x(R, S)) + \mathbb{I}_{IDX}(R, S) \cdot \text{RMPTCOST}(rows(R) \cdot H(I_S), x(R))$
Hash join (HJ)	$C_{HJ}(R, S) = C_{HJ}^{BASE}(R, S) + \text{DATAMOVECOST}(ws^{HJ}(S), x_{HJ}(S)) + \text{RMPTCOST}(a_{HJ}(R, S), x_{HJ}(S))$	Merge join (MJ)	$C_{MJ}(R, S) = C_{MJ}^{BASE}(R, S) + \text{DATAMOVECOST}(ws^{MJ}(R, S), x(R, S)) + \mathbb{I}_{IDX}(R, S) \cdot \text{RMPTCOST}(a_{MJ}(R, S), x(R, S))$
Materialize (MAT)	$C_{MAT}(R) = C_{MAT}^{BASE}(R) + \text{DATAMOVECOST}(ws^{MAT}(R), x(R))$	Memoize (MEM)	$C_{MEM}(R) = C_{MEM}^{BASE}(R) + \text{DATAMOVECOST}(ws^{MEM}(R), x(R))$

new cost model, summarized in Table 2. For each operator, beyond its base cost C_{Op}^{BASE} capturing conventional CPU and I/O activities, we add CVM-aware overhead terms determined by its input relation R and, when applicable, the other input relation S and index I .

Working Set and CVM Cost Primitives. A key concept in our model is the *working set* (ws), the amount of data that must remain *active* during an operator’s execution phase. We derive ws from optimizer-visible statistics: for base-table relations, memory usage is page-oriented, so we approximate ws from *pages* via a byte-level translation; for intermediate results, it is tuple-oriented, so we approximate it as $rows \times width$. Once ws exceeds cache capacity, more encrypted memory traffic, bounce-buffer interaction, and spill-related movement would be triggered. To capture this excess, we define an overflow factor $x = \max(0, (ws - (1 + \epsilon)C)/C)$, where C is the effective cache budget and ϵ is a grace factor introducing a tolerance band near the cache boundary to reduce instability from small estimation errors. We then model expensive data movement with the *data-movement cost term* $\text{DATAMOVECOST}(ws, x) = ws \cdot \eta_{DM} \cdot \sigma(x)$, where η_{DM} is a calibrated per-unit penalty for ws and $\sigma(x)$ activates the term in the overflow regime. The overflow-related parameters and η_{DM} should be set so that the term reacts primarily when the active footprint can no longer remain cache-resident. We also model translation and integrity-validation overhead with the *RMP-related cost term* $\text{RMPTCOST}(a, x) = a \cdot \eta_{RMP} \cdot (1 + \alpha_{RMP} \cdot \sigma(x))$, where a captures page-access intensity and locality, η_{RMP} is the calibrated per-access penalty for a , and α_{RMP} raises the cost under overflow. This follows the second motivation result: even under the same plan, CVM increases cycles and CPI, suggesting the dominant effect is not more work but higher latency along address-translation and protected memory-access paths. We tune these two terms’ parameters on a representative subset of queries from each workload; details are in Section 4. Our goal is not exhaustive optimization, but to show that a reasonable cost-model calibration can reduce the performance gap between KVM and CVM.

Scan Operators. *Sequential scans (SS)* exhibit strong spatial locality and predictable streaming access. Their principal CVM vulnerability is therefore not pointer chasing, but the growth of the active scan footprint. When ws exceeds cache capacity, memory traffic increasingly flows through encrypted paths and amplifies bounce-buffer-related overheads. Accordingly, we preserve the base sequential-scan cost and add a DATAMOVECOST term. *Bitmap scans*

and related heap-fetch operators follow the same logic: their main CVM sensitivity is overflow-driven data movement, so they are also charged via DATAMOVECOST .

Index-only and index-driven scans differ because index-based traversal involves pointer chasing and many small, often non-sequential page accesses. These accesses increase translation intensity and are more aligned with the CPI-inflation evidence in Table 1. If a B-tree is used as the index, then the estimated height is $H(I) = \max\{1, \log_F(\text{PAGES}(I))\}$, where F denotes the effective fanout implied by the index structure and page layout. We then use $rows_{OUT}(R) \cdot H(I)$ as the proxy in RMPTCOST , since each returned tuple may trigger a descent across multiple index tree levels. If the accessed data by the index also becomes large, the operator may additionally incur DATAMOVECOST .

Join Operators. Join algorithms expose CVM overheads differently. *Nested-loop joins (NL)* repeatedly probe the inner side for each outer tuple. When the inner side is index-driven, NL can trigger many small and repeated index descents, which raise address translation pressure; hence we add a conditional RMPTCOST term. Repeated probing and possible buffering enlarge the effective active footprint, so NL also accrues DATAMOVECOST when overflow occurs.

Merge joins (MJ) are more sequential. Their base behavior is dominated by streaming reads and, if needed, sorting, so their primary CVM exposure is overflow-driven data movement. Still, if one input is produced by an index-heavy plan with weaker locality, translation overhead may also matter, which is why Table 2 includes a conditional RMPTCOST term.

Hash joins (HJ)’s calibration reflects the clearest working-set phase change. When the build-side hash table fits in the last-level cache (LLC, i.e., the processor’s shared L3 cache), performance remains relatively stable; once it exceeds the LLC threshold W_{L3} determined by the system, the operator becomes more sensitive to encrypted memory traffic and random accesses. This behavior is consistent with the motivation evidence that different join strategies induce different data-movement patterns in SEV-SNP. We therefore define $x_{HJ} = \max(0, HT(S)/W_{L3} - 1)$, where $HT(S)$ is the estimated hash-table size. HJ then incurs both DATAMOVECOST for overflow-driven movement and RMPTCOST for address translation pressure.

Other Operators. *Materialize* and *Memoize* operators primarily create or retain intermediate state. Their dominant CVM sensitivity is again governed by the size of the active footprint rather than by

Table 3: The calibration’s time savings in SEV-SNP.

Workload	Baseline (s)	with Calibration (s)	Time Savings
JOB	318.63	296.63	6.90%
CEB	780.51	646.71	17.14%
Stack	354.72	183.20	48.35%
TPC-DS	4049.78	3906.28	3.54%

complex pointer chasing. Once the buffered state exceeds the cache capacity, encrypted data movement and bandwidth pressure rise. We therefore augment their base costs with $\text{DATAMOVECOST}(ws, x)$; memoization may also indirectly affect downstream locality by changing later operators’ working sets and access patterns.

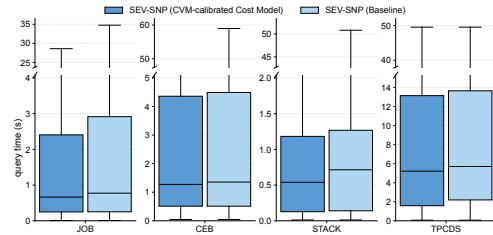
4 Preliminary Evaluation

In this section, we evaluate how much time savings the light-weight CVM-aware calibration can achieve and what fraction of queries benefit from it. Finally, we examine the latency distribution before and after applying the cost model calibration.

Experimental Setup. We evaluate the calibration on TPC-DS [8] (scale factor 10, two instances for each of the 99 query templates), Stack [30] (10 instances for each of the 16 templates), and IMDB-based [23] JOB [23] (all 113 queries) and CEB [37] (10 instances for each of the 16 templates). We set the timeout to 60s. We execute the queries in a sequential and reproducible way. For each query, we collect its better performance from enable/disable the calibration. We implemented the calibration in PostgreSQL 16.9. Its default optimizer is our baseline. We use an adapted PostgreSQL [9] to support C extension design easily. A default parameter set (η_{DM} , η_{RMP} , etc.) is used in all evaluations. Starting from random initial values, we tune one parameter at a time while fixing the others, and retain settings that perform robustly across all workloads using one-third randomly sampled queries per workload. We evaluate the same virtual machine configuration under two security levels: no protection (KVM) and SEV-SNP. The host machine has 128 GiB RAM and an AMD EPYC 7413 CPU. We configure SEV-SNP/KVM with 64 GiB memory and 32 vCPUs.

Main Evaluation. As shown in Table 3, our calibration reduces total workload running time by a range of 3.54% to 48.35%. This result highlights a key strength of the CVM-calibrated cost model: calibrating the optimizer’s cost model can steer DBMS toward substantially better plans under CVM overhead. For vanilla PostgreSQL in KVM, whose execution times are 259.30 s (JOB), 677.16 s (CEB), 329.53 s (Stack), and 3829.20 s (TPC-DS), using cost-model calibration in SEV-SNP surpasses the KVM baseline on CEB and Stack, indicating that the calibration effectively bridges a significant portion of the CVM-induced performance gap.

The calibration improves a substantial fraction of queries across all workloads: 43.4% in JOB, 38.8% in CEB, 62.5% in Stack, and 44.4% in TPC-DS, showing that the benefit of it is broad. The improvement ratio nonetheless varies across workloads, suggesting that workload characteristics affect both how often and how much a fixed calibration helps. One likely reason is that different workloads expose different optimizer weaknesses. For workloads whose performance is highly sensitive to operator choice and relative operator costing, a better CVM-aware cost model can more directly correct plan

**Figure 3: The different query latency distributions.**

selection and yield larger savings. In contrast, when the dominant source of error is cardinality estimation, improving operator costs alone has a limited impact. The gap may also reflect differences in plan structure, memory pressure, etc. These observations point to a direction for future work: combining CVM-aware cost calibration with workload-adaptive parameter tuning, and integrating it more tightly with other DBMS components and data properties, has the potential to deliver stronger end-to-end gains.

Latency Distribution. We next examine the query latency distribution for each workload in SEV-SNP, comparing vanilla PostgreSQL with the calibration in Figure 3. It shifts the distribution downward by lowering the median and compressing the spread. The effect is most striking on Stack, where the median drops from 0.71 s to 0.53 s, the standard deviation shrinks from 7.00 s to 2.24 s, and the maximum latency falls sharply from 50.80 s to 13.84 s. CEB shows a similar long-tail reduction, with the maximum 28.12 s decrease. JOB also improves more broadly, with both median latency reduced (0.77 s to 0.66 s) and the tail shortened (34.77 s to 28.60 s). The calibration corrects the optimizer’s cost reasoning under CVM overhead and helps PostgreSQL avoid fragile plans in SEV-SNP.

5 Related Work

Prior work on DBMSs under TEEs mainly spans four directions. First, existing TEE-based DBMSs [13, 14, 19, 24, 43] focus on security architecture and trusted execution. Second, operator-level studies design oblivious joins, sorting, and related primitives with formal leakage protection [18, 22, 27, 31]. Third, benchmarking work shows that trusted execution can substantially slow analytical workloads [6, 25, 26, 28]; more broadly, a growing line of work benchmarks CVMs against KVM baselines and studies how to optimize CVM execution beyond DBMSs [20, 32, 46]. Finally, our work also relates to database research on adapting cost models to hardware characteristics [7, 29]. Unlike prior TEE database work, which mainly emphasizes secure execution or overhead measurement, we target the underexplored optimizer side: improving database performance in CVMs through lightweight cost-model calibration.

6 Conclusion

We present a light-weight CVM-aware calibration for the cost model and integrate it into PostgreSQL. Our design augments a default cost modeling in the query optimizer with simple penalties that better reflect CVM-specific overheads: data movement and RMP checks. This calibration can improve query planning decisions and reduce execution time in SEV-SNP, which mitigates the performance gap between KVMs and CVMs.

References

- [1] Advanced Micro Devices, Inc. *AMD SEV Secure Nested Paging Firmware ABI Specification*, 2025. Document 56860, Rev. 1.58 (May 2025).
- [2] Ayaz Akram, Anna Giannakou, Venkatesh Akella, Jason Lowe-Power, and Sean Peisert. Performance analysis of scientific computing workloads on general purpose tees. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1066–1076. IEEE, 2021.
- [3] AMD. Confidential computing. <https://www.amd.com/en/products/processors/server/epyc/confidential-computing.html>, 2025. Accessed: 2025-12-25.
- [4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.
- [5] Syafiq Al Attiq and Aris Cahyadi Risdianto. Demystifying amd sev performance penalty for nfv deployment. In *Proceedings of the 2024 13th International Conference on Networks, Communication and Computing*, pages 1–8, 2024.
- [6] Ilaria Battiston, Lotte Felius, Sam Ansmink, Laurens Kuiper, and Peter Boncz. Duckdb-sgx2: The good, the bad and the ugly within confidential analytical query processing. In *Proceedings of the 20th International Workshop on Data Management on New Hardware*, pages 1–5, 2024.
- [7] Daniel Bausch, Ilia Petrov, and Alejandro Buchmann. Making cost-based query optimization asymmetry-aware. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, pages 24–32, 2012.
- [8] TPC-DS Benchmark. <https://www.tpc.org/tpcds/>.
- [9] Rico Bergmann, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. An elephant under the microscope: Analyzing the interaction of optimizer components in postgresql. *Proceedings of the ACM on Management of Data*, 3(1):1–28, 2025.
- [10] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90, 2009.
- [11] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.
- [12] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.
- [13] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *arXiv preprint arXiv:1710.00458*, 2017.
- [14] Benny Fuhry, HA Jayanth Jain, and Florian Kerschbaum. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 438–450. IEEE, 2021.
- [15] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K Cunningham. Sok: Cryptographically protected database search. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 172–191. IEEE, 2017.
- [16] Goetz Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.
- [17] Goetz Graefe and William J McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of IEEE 9th international conference on data engineering*, pages 209–218. IEEE, 1993.
- [18] Tianyao Gu, Yilei Wang, Afonso Tinoco, Bingnan Chen, Ke Yi, and Elaine Shi. Flexway {O-Sort} : {Enclave-Friendly} and optimal oblivious sorting. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 7563–7582, 2025.
- [19] Ziyang Han and Haibo Hu. Prodb: A memory-secure database using hardware enclave and practical oblivious ram. *Information Systems*, 96:101681, 2021.
- [20] Benjamin Holmes, Jason Waterman, and Dan Williams. Severifast: Minimizing the root of trust for fast startup of sev microvms. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 1045–1060, 2024.
- [21] Intel. Intel® trust domain extensions (intel® tdx) overview. <https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html>, 2025. Accessed: 2025-12-25.
- [22] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. Efficient oblivious database joins. *arXiv preprint arXiv:2003.09481*, 2020.
- [23] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.
- [24] Mingyu Li, Xuyang Zhao, Le Chen, Cheng Tan, Huorong Li, Sheng Wang, Zeyu Mi, Yubin Xia, Feifei Li, and Haibo Chen. Encrypted databases made secure yet maintainable. In *17th USENIX symposium on operating systems design and implementation (OSDI 23)*, pages 117–133, 2023.
- [25] Adrian Lutsch, Muhammad El-Hindi, Matthias Heinrich, Daniel Ritter, Zsolt Istváň, and Carsten Binnig. Benchmarking analytical query processing in intel sgxv2. *arXiv preprint arXiv:2403.11874*, 2024.
- [26] Adrian Lutsch, Christian Franck, Muhammad El-Hindi, Zsolt Istváň, and Carsten Binnig. An analysis of aws nitro enclaves for database workloads. In *Proceedings of the 21st International Workshop on Data Management on New Hardware*, pages 1–8, 2025.
- [27] Kajetan Maliszewski, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. Cracking-like join for trusted execution environments. *Proceedings of the VLDB Endowment*, 16(9):2330–2343, 2023.
- [28] Kajetan Maliszewski, Jorge-Arnulfo Quiané-Ruiz, Jonas Traub, and Volker Markl. What is the price for joining securely? benchmarking equi-joins in trusted execution environments. *Proceedings of the VLDB Endowment*, 15(3):659–672, 2021.
- [29] Stefan Manegold, Peter Boncz, and Martin L Kersten. Generic database cost models for hierarchical memory systems. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 191–202. Elsevier, 2002.
- [30] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *SIGMOD*, pages 1275–1288, 2021.
- [31] Apostolos Mavrogiannakis, Xian Wang, Ioannis Demertzis, Dimitrios Papadopoulos, and Minos Garofalakis. Oblivator: Oblivious parallel joins and other operators in shared memory environments. *Cryptology ePrint Archive*, 2025.
- [32] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. Confidential vms explained: An empirical analysis of amd sev-snp and intel tdx. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8(3):1–42, 2024.
- [33] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. Confidential vms explained: An empirical analysis of amd sev-snp and intel tdx. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8(3):1–42, 2024.
- [34] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. A comparison study of intel sgx and amd memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8, 2018.
- [35] Dominic P Mulligan, Gustavo Petri, Nick Spinale, Gareth Stockwell, and Hugo JM Vincent. Confidential computing—a brave new world. In *2021 international symposium on secure and private execution environment design (SEED)*, pages 132–138. IEEE, 2021.
- [36] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.
- [37] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. Flow-loss: Learning cardinality estimates that matter. *Proc. VLDB Endow.*, 14(11):2019–2032, jul 2021.
- [38] PostgreSQL. <https://www.postgresql.org/>.
- [39] Lina Qiu, Rebecca Taft, Alexander Shraer, and George Kollios. The price of privacy: a performance study of confidential virtual machines for database systems. In *Proceedings of the 20th International Workshop on Data Management on New Hardware*, pages 1–8, 2024.
- [40] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, 1979.
- [41] AMD Sev-Snp. Strengthening vm isolation with integrity protection and more. *White Paper, January*, 53(2020):1450–1465, 2020.
- [42] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1041–1052, 2017.
- [43] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. Stealthdb: a scalable encrypted database with full sql query support. *Proceedings on Privacy Enhancing Technologies*, 2019.
- [44] Haoyu Wang, Noa Zilberman, Ahmad Atamli, and Amro Awad. Revisiting virtual memory support for confidential computing environments, 2025.
- [45] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. Security: No security without integrity: Breaking integrity-free memory encryption with minimal assumptions. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1483–1496. IEEE, 2020.
- [46] Mingjie Yan and Kartik Gopalan. Performance overheads of confidential virtual machines. In *2023 31st International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 1–8. IEEE, 2023.