E-LQO: A Comprehensive Energy Evaluation Framework for Learned Query Optimization

ZIBO LIANG, University of Electronic Science and Technology of China, China
QUANQING XU*✉, Independent Researcher, China
XU CHEN, University of Electronic Science and Technology of China, China
JUNMING CHEN, University of Electronic Science and Technology of China, China
YUYANG XIA, University of Electronic Science and Technology of China, China
KAI ZHENG*✉, University of Electronic Science and Technology of China, China

Learned Query Optimization (LQO) has emerged as a promising paradigm for improving database performance, with reported speedups of 2× to over 10× on standard benchmarks. Existing evaluations, however, remain largely latency-centric and leave the energy debt from data collection, model training, and online inference less visible. This paper presents E-LQO, the first comprehensive framework for evaluating energy consumption across the complete LQO lifecycle: data collection or preparation, model training, model inference, and plan execution. We formalize two domain-specific metrics—*Energy Return on Investment* (EROI) and *Energy Payback Time* (EPT)—to quantify both per-cycle efficiency and the amortization horizon of learned optimizers. Across seven LQO methods on fixed-scale JOB and on TPC-H/TPC-DS up to 100 GB, we find that lifecycle energy is governed less by whether an optimizer is learned than by how it obtains supervision. Execution-based systems buy plan quality through active exploration and carry a large upfront energy debt; passive/log-driven systems can repay much faster with reusable logs or labels, but this advantage depends on incremental-cost accounting and often comes with weaker latency gains on light workloads. We further show that total online inference energy, including candidate-plan generation, enables symmetric accounting, and that roughly 50% template coverage is the energy-optimal exploration region for the execution-based methods we study. E-LQO provides practitioners with actionable guidance for energy-conscious deployment decisions and exposes optimization opportunities for sustainable learned query optimization.

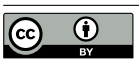
CCS Concepts: • **Information systems** → **Query optimization**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Database, Energy-efficient system, Query processing and optimization

ACM Reference Format:
Zibo Liang, Quanqing Xu, Xu Chen, Junming Chen, Yuyang Xia, and Kai Zheng. 2026. E-LQO: A Comprehensive Energy Evaluation Framework for Learned Query Optimization. *Proc. ACM Manag. Data* 4, 4 (SIGMOD), Article 269 (September 2026), 27 pages. https://doi.org/10.1145/3837107

*Quanqing Xu and Kai Zheng are co-corresponding authors.

Authors' Contact Information: Zibo Liang, zbliang@std.uestc.edu.cn, University of Electronic Science and Technology of China, Chengdu, Sichuan, China; Quanqing Xu, xuquanqing@gmail.com, Independent Researcher, Hangzhou, Zhejiang, China; Xu Chen, XUCHEN.2019@outlook.com, University of Electronic Science and Technology of China, Chengdu, Sichuan, China; Junming Chen, junmingchen@std.uestc.edu.cn, University of Electronic Science and Technology of China, Chengdu, Sichuan, China; Yuyang Xia, xiayuyang@std.uestc.edu.cn, University of Electronic Science and Technology of China, Chengdu, Sichuan, China; Kai Zheng, zhengkai@uestc.edu.cn, University of Electronic Science and Technology of China, Chengdu, Sichuan, China.



ACM 2836-6573/2026/9-ART269
https://doi.org/10.1145/3837107

1 Introduction

Query optimization stands as one of the most critical components in modern database management systems (DBMSs), directly determining the efficiency of query execution. In recent years, Learned Query Optimization (LQO) has emerged as a promising paradigm that leverages machine learning to enhance or replace traditional optimizer components. The field is heterogeneous: execution-based designs such as Neo [37] and Bao [36] actively explore plans, while passive/log-driven designs such as MSCN [20] learn from existing query-cardinality records. We evaluate Bao, Balsa, and LEON as execution-based systems and compare them with passive representatives later. Subsequent research has further explored learned cost models and reinforcement learning strategies. These approaches have demonstrated substantial speedups, often ranging from 2× to over 10× on benchmarks such as JOB [28] and TPC-H [2], fostering a growing consensus that learning-based optimization represents the natural evolution toward more intelligent database systems.

Despite these achievements, recent benchmarking efforts have scrutinized the robustness of LQO methods [48], revealing that learned optimizers may exhibit performance degradation under workload shifts, data distribution changes, or unseen query patterns [27]. Nevertheless, within standard experimental settings, the efficacy of LQO remains well-established and broadly acknowledged. A closer examination of existing benchmarks, however, uncovers a critical blind spot: nearly all evaluations adopt time-centric metrics—such as query latency, model inference time, or end-to-end execution time—as the primary, if not sole, measure of success. While latency is undoubtedly important, this narrow focus overlooks a metric of growing significance in modern data centers: *energy consumption*.

As cloud-based database services continue to scale, energy efficiency has become a first-class concern for both economic and environmental reasons [22, 55]. Training machine learning models, executing exploratory query plans for data collection, and performing repeated inference all impose substantial computational overhead that translates directly into electricity consumption [57]. Yet, to the best of our knowledge, no prior work has systematically quantified the energy footprint of LQO across its full lifecycle. Execution-based and passive/log-driven LQO form distinct cost profiles with different latency–energy tradeoffs [52]. This raises a fundamental question: *when accounting for energy costs, do the performance gains of learned query optimization justify the overhead introduced by its machine learning components, and under which LQO paradigm?* In this paper, we aim to bridge this gap by proposing a comprehensive energy-aware evaluation framework for LQO, shifting the lens from pure latency optimization to a more holistic cost-benefit analysis. Specifically, we formulate and investigate three key Research Questions (RQs).

RQ1: Where does the energy go throughout the LQO lifecycle? To rigorously assess the energy efficiency of LQO, we must first understand how energy is distributed across its operational lifecycle. Unlike traditional query optimizers that incur costs solely during query compilation and execution, LQO introduces additional energy-consuming phases: *data collection*, where execution-based systems explore query plans beyond the default optimizer's choices to gather training samples (or, for passive methods, where reusable query logs and cardinality labels are prepared for training); *model training*, where neural networks or other learnable components are updated based on collected experiences; *model inference*, where the trained model predicts costs or selects plans for incoming queries; and *plan execution*, where the chosen plans are physically executed. As illustrated in Figure 1b, the LEON lifecycle breakdown [5] reveals a striking finding: data collection alone accounts for 93–98% of the total energy investment across the evaluated benchmarks (IMDb, TPC-H, and TPC-DS) for execution-based LQO. This disproportionate energy consumption stems from the necessity of executing numerous suboptimal or exploratory plans to provide sufficient training signals. For passive/log-driven methods, this cost shifts from active exploration to preparation and

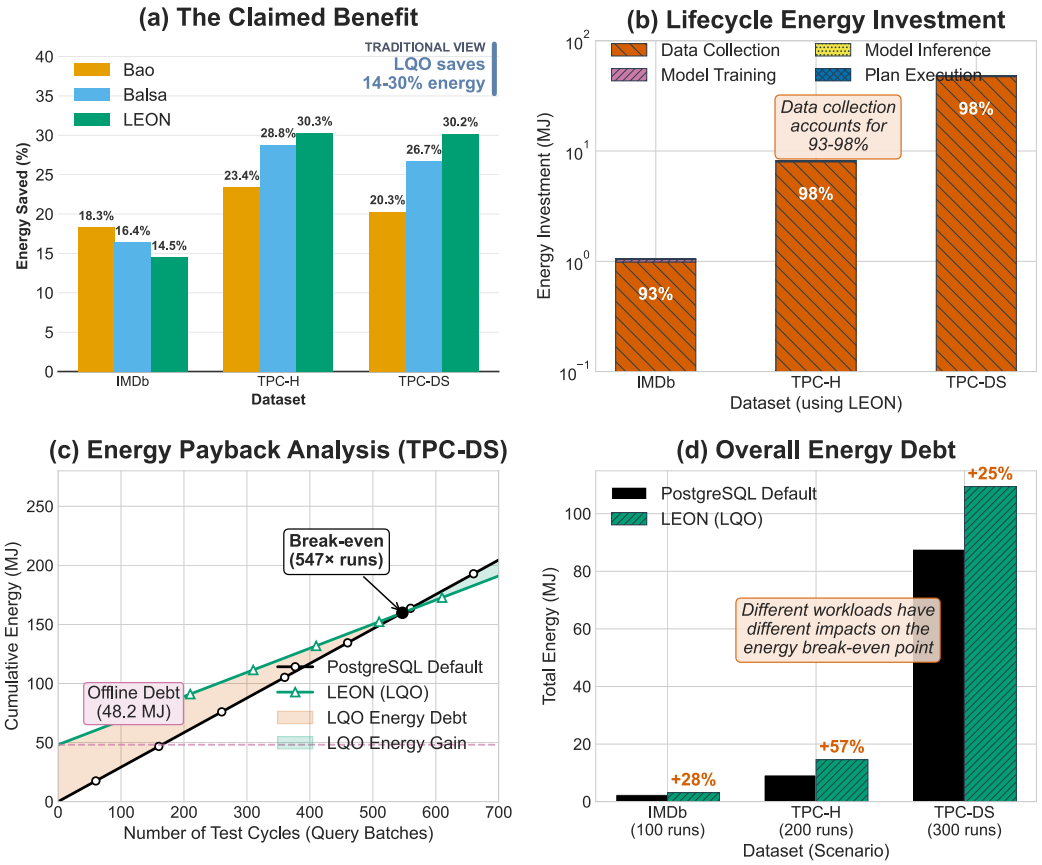


Fig. 1. An energy-centric perspective on Learned Query Optimization. Per-method measurements appear in Section 5.2–5.6.

model training when reusable logs or labels are available. Identifying such energy bottlenecks is essential for guiding future optimization efforts.

RQ2: When does LQO achieve energy break-even? From a conventional latency-centric perspective, LQO methods demonstrate compelling benefits. As shown in Figure 1a, representative approaches such as Bao, Balsa, and LEON achieve 14–30% online energy savings compared to the PostgreSQL default optimizer under the unified $E_{\text{infer}}^{\text{total}}$ accounting (Eq. 2). However, this optimistic view considers only the operational (online) phase while neglecting the substantial offline energy debt incurred before deployment. When we account for the full lifecycle costs, a different picture emerges. Figure 1c presents a fixed-scale energy payback analysis on TPC-DS, where we plot cumulative energy consumption against the number of test cycles (each cycle represents one complete execution of the test query batch). Despite LEON's lower learned-plan execution energy, the 48.2 MJ offline energy debt creates an initial energy deficit that requires several hundred test cycles to amortize—equivalent to executing the entire test workload several hundred times before any net energy benefit materializes. Moreover, as illustrated in Figure 1d, the number of test cycles required to reach energy break-even varies considerably across different datasets and workload characteristics. Beyond this fixed-scale payback example, our 100 GB realistic-buffer analysis reports

full online test-cycle energy including inference and CPU/DRAM effects: active methods still need hundreds of cycles, while passive methods repay quickly with reusable supervision. These findings underscore that energy-aware evaluation must encompass both training and inference phases, and reveal that the energy benefits of LQO are highly workload-dependent and paradigm-dependent.

RQ3: Is LQO more economical than hardware scaling? The deployment of LQO often uses additional computational resources—especially GPUs for model training, and either CPUs or GPUs for inference—which introduce their own energy overhead beyond the baseline database workload. In the era of cloud computing, an alternative strategy for improving query performance is straightforward hardware scaling—provisioning more powerful CPUs, additional memory, or faster storage. This raises a pragmatic question: from a pure energy perspective, does investing in LQO yield better returns than simply allocating equivalent energy budgets to enhanced hardware? We compare PostgreSQL, Bao, Balsa, and LEON under the same serial, 6-connection parallel, and 12-connection scaled configurations, and report CPU/GPU inference sensitivity.

To address these research questions, we design and implement E-LQO, the first comprehensive energy evaluation framework specifically tailored for learned query optimization. Our framework instruments the complete LQO lifecycle—spanning data collection, model training, model inference, and plan execution—with fine-grained energy measurement capabilities. By decomposing energy consumption across these four distinct phases, E-LQO enables practitioners and researchers to pinpoint energy bottlenecks, quantify the true cost-benefit tradeoffs of adopting LQO, and make informed decisions about when and where learned optimization is genuinely worthwhile. We conduct extensive experiments on seven LQO methods across fixed-scale IMDb/JOB (3.6 GB) and TPC-H/TPC-DS at 10–100 GB under memory-resident and realistic-buffer settings, revealing insights that challenge the prevailing latency-centric narrative and offer practical guidance for energy-conscious database system design.

In summary, this paper makes the following contributions:

- We propose E-LQO, the first framework to systematically measure and analyze energy consumption across the full lifecycle of learned query optimization, covering data collection, model training, model inference, and query plan execution.
- We formalize the first set of domain-specific metrics for green query optimization: Energy ROI (EROI) and Energy Payback Time (EPT). The metrics capture both online energy efficiency and lifecycle amortization, allowing practitioners to identify when learned optimization yields net energy benefits and when its overhead outweighs its gains. These metrics provide a standard language for future research to benchmark sustainability.
- We characterize Bao, Balsa, and LEON under memory-resident, realistic-buffer, and concurrent settings, showing when scale helps and when hardware parallelism erodes the advantage.
- We extend E-LQO to NeuroCard, MSCN, Zero-shot, and DACE, quantifying passive/log-driven lifecycle energy under incremental-cost deployment and comparing it directly with execution-based exploration.
- We show that the payback-optimal training-template coverage ratio α^*—the fraction of query templates used for exploration—lies near one half ($\alpha^* \in [0.46, 0.57]$), model architecture changes payback, and passive designs trade fast payback for weaker latency gains on light workloads.

2 Background

This section introduces the representative learned query optimization methods (§2.1), surveys learned cardinality and cost estimation techniques (§2.2), and reviews existing work on energy-aware database systems (§2.3).

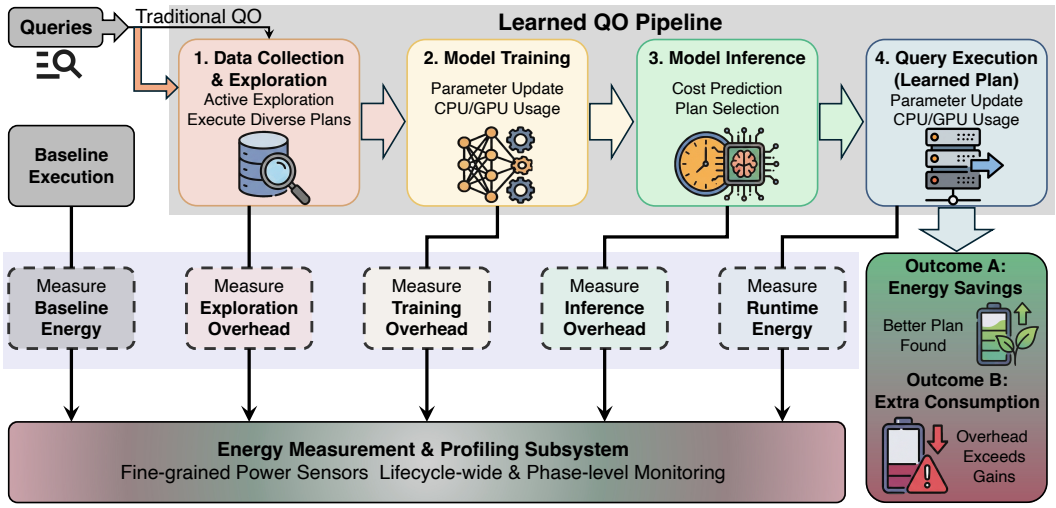


Fig. 2. Framework Overview of E-LQO.

2.1 Learned Query Optimization

Learned Query Optimization (LQO) methods differ fundamentally in their *plan exploration space* and *integration strategy* [5, 31, 36, 37, 41, 69, 73]. We focus on three representative execution-based methods: Bao [36], Balsa [69], and LEON [5], while passive/log-driven LQO (§2.2) represents a complementary class of methods.

Bao [36] operates within a constrained hint space. Rather than generating plans from scratch, Bao learns to select optimizer hints (e.g., disabling hash joins, forcing index scans) that steer the underlying optimizer toward better choices. The hint space is limited to 25 distinct configurations, ensuring plan validity while sacrificing potential gains from plans outside the hint-reachable region.

Balsa [69] takes an ambitious stance by entirely replacing the traditional optimizer with a reinforcement learning agent. The agent learns to construct query plans by selecting join orders [53] and physical operators [38] without relying on expert demonstrations. The exploration space encompasses all possible join orderings along with a customizable operator set, offering potential for discovering novel strategies but requiring extensive training.

LEON [5] augments the traditional optimizer by training a pairwise ranking model that compares candidate plans. Given a query, LEON leverages the traditional optimizer to generate a set of candidate plans, then employs the learned ranker to correct the optimizer's cost estimates and identify plans that may outperform the default choice. Similarly, Lero [75] addresses the problem using classification-based cost correction. The three methods span a clear exploration-breadth ordering (LEON > Balsa > Bao) that directly determines the dominant data-collection energy term we quantify in Section 5.2.

2.2 Learned Cardinality and Cost Estimation

Cardinality estimation has long been the Achilles' heel of cost-based optimization: classical histogram-based estimators [17] can deviate by orders of magnitude from actual cardinalities, and even small errors propagate super-linearly into plan-cost predictions [43]. These limitations motivated the family of learned estimators surveyed below. We distinguish cardinality estimators,

plan-level cost models, and passive/log-driven LQO because they induce different E-LQO lifecycle costs.

Learned cardinality estimators such as MSCN [20] and NeuroCard [70] (an autoregressive joint distribution over table attributes [9]) feed learned cardinalities back into the traditional optimizer. With reusable query–cardinality records or table samples, these methods avoid active plan exploration and therefore induce a different lifecycle energy profile from execution-based LQO.

Plan-level cost models include Tree-LSTM-based models [15, 58, 60], Transformer [62]-based QueryFormer [72], Stage [65], Zero-shot [14], and DACE [33]. Some consume learned cardinalities, so the upstream cardinality module is included; others bypass a separate cardinality module and learn directly from plan-tree features or logged plan–latency records [13]. We evaluate these methods under incremental-cost accounting when such logs or records are available.

Passive/log-driven LQO (e.g., JGMP [52]) learns from query logs instead of active exploration. Section 5.6 uses NeuroCard, MSCN, Zero-shot, and DACE to quantify when this supervision makes payback fast and when model cost still dominates; Section 5.5 reports backbone sensitivity within the execution-based family.

2.3 Energy Efficiency in Database Systems

Energy consumption has emerged as a critical concern in computing, driven by economic pressures and environmental sustainability. Data centers account for approximately 1–2% of global electricity consumption [7, 39], with database systems representing significant energy consumers due to their computationally intensive, continuous operations.

Foundational Work. Barroso and Hölzle [3] introduced energy proportionality—the principle that systems should consume power proportional to utilization—establishing the conceptual foundation for subsequent research. Harizopoulos et al. [12] argued that software optimization is essential for energy efficiency, articulating the database community's green research agenda.

Energy-Aware Query Processing. Tsirogiannis et al. [61] conducted pioneering measurements of query energy consumption, establishing that different operators exhibit distinct energy profiles—CPU power varies by up to 60% across operators at identical utilization. Lang and Patel [25] first treated energy as a query processing optimization goal, demonstrating 49% processor energy reduction with minimal performance impact through DVFS techniques [26, 54]. Xu et al. [67] proposed PET, integrating energy considerations directly into PostgreSQL's query optimizer.

System-Level Design. Lang et al. [24] examined energy-efficient parallel DBMS cluster design, comparing high-performance versus low-power node configurations. Korkmaz et al. [23] presented POLARIS for DBMS-controlled DVFS [21, 50], achieving approximately 40W savings while meeting latency targets. Concurrency-aware scheduling and cloud resource-management studies [6, 32, 47, 51, 63, 66, 71, 74] motivate our serial/parallel/scaled comparison against hardware scaling in Section 5.4.

The Gap: AI Component Energy. Despite this body of work, a notable gap exists regarding energy consumption of machine learning components within database systems. As surveyed by Guo et al. [11], existing research focuses exclusively on traditional query processing. The energy footprint of learned components—training data collection, model training, and inference—remains largely unquantified. This gap is particularly consequential for LQO, where ML is integral to query processing rather than a one-time preprocessing step. Our work addresses this gap by providing the first comprehensive energy evaluation framework specifically designed for learned query optimization, spanning both execution-based (Bao/Balsa/LEON) and passive/log-driven (NeuroCard/MSCN/Zero-shot/DACE) families.

3 Overview

This section formalizes the energy evaluation problem for learned query optimization (§3.1) and presents an overview of the E-LQO framework (§3.2).

3.1 Problem Formulation

We formalize the energy consumption evaluation problem for Learned Query Optimization (LQO) by decomposing the total energy footprint across distinct operational phases. Let $Q = \{q_1, q_2, \ldots, q_n\}$ denote a workload [34] consisting of n queries. The total energy consumption of an LQO system can be expressed as:

$$E_{\text{LQO}}^{\text{total}} = E_{\text{collect}} + E_{\text{train}} + E_{\text{infer}} + E_{\text{exec}} \tag{1}$$

where each term corresponds to a distinct phase in the LQO lifecycle. For passive/log-driven methods, the first term is denoted E_{prep} in the experiments but occupies the same accounting slot as E_{collect} in Eq. 1.

We use two metrics, defined formally below: Energy Return on Investment (EROI) measures per-cycle efficiency, while Energy Payback Time (EPT) measures the number of test-workload cycles needed to amortize offline energy.

- E_{collect}: **Data Collection Energy.** The energy consumed during the exploration phase, where the system executes diverse query plans beyond the default optimizer's choices to gather training samples. This phase typically dominates the training-phase energy consumption, as it requires executing numerous suboptimal or exploratory plans. For passive/log-driven methods (Section 5.6), this term is reinterpreted as E_{prep} (data preparation), covering additional feature-extraction or sample-scan energy over reusable query logs.
- E_{train}: **Model Training Energy.** The energy consumed for updating the parameters of neural networks or other learnable components based on the collected experiences. This includes both CPU and GPU computational overhead during gradient computation and weight updates.
- E_{infer}: **Model Inference Energy.** The energy consumed when the trained model predicts costs or selects plans for incoming queries during the online serving phase. For both metrics, this term is aggregated over the test workload: $E_{\text{infer}} = \sum_{q_i \in Q_{\text{test}}} e_{\text{infer}}^{(i)}$.
- E_{exec}: **Plan Execution Energy.** The energy consumed during the physical execution of the selected query plans. For LQO, this corresponds to executing the learned plans over the test workload: $E_{\text{exec}} = \sum_{q_i \in Q_{\text{test}}} e_{\text{exec}}^{(i)}$, where $e_{\text{exec}}^{(i)}$ denotes the execution energy for query q_i.

Inference-energy accounting convention. When computing either metric for one complete test-workload cycle, E_{infer} denotes total online optimization overhead:

$$E_{\text{infer}}^{\text{total}} = E_{\text{infer}}^{\text{plan\_gen}} + E_{\text{infer}}^{\text{feature\_encode}} + E_{\text{infer}}^{\text{model\_forward}}, \tag{2}$$

covering candidate enumeration, plan-tree encoding, and model evaluation. PostgreSQL planner activity stays inside the baseline cycle; passive/log-driven methods are accounted for end to end because their learned estimates are consumed by the conventional enumerator.

To assess whether LQO provides energy benefits, we compare against the baseline energy consumption of a traditional query optimizer. The baseline energy is defined as:

$$E_{\text{baseline}} = \sum_{q_i \in Q_{\text{test}}} e_{\text{baseline}}^{(i)} \tag{3}$$

where $e_{\text{baseline}}^{(i)}$ represents the energy consumed when executing query q_i using the plan selected by the traditional optimizer (e.g., PostgreSQL's default planner). Unless otherwise stated, E_{baseline}, E_{exec}, and E_{infer} in both metrics all refer to one complete test-workload cycle.

We further introduce two domain-specific metrics to quantify the energy efficiency of LQO:

Energy Return on Investment (EROI) [45]. This metric quantifies the energy conversion efficiency during the inference phase by contrasting the execution energy savings against the inference overhead:

$$\text{EROI} = \frac{E_{\text{baseline}} - E_{\text{exec}}}{E_{\text{infer}}} \tag{4}$$

An EROI greater than 1 indicates that the energy saved through improved plan selection exceeds the inference overhead. Higher EROI values signify more efficient utilization of inference energy.

Energy Payback Time (EPT) [45]. This metric measures the amortization horizon, specifically the number of query cycles k required to offset the initial training energy debt:

$$\begin{aligned}\text{EPT} = \min\{k \in \mathbb{Z}^+ \mid k \cdot (E_{\text{baseline}} - E_{\text{exec}} - E_{\text{infer}}) \\ \geq E_{\text{collect}} + E_{\text{train}}\}\end{aligned} \tag{5}$$

EPT captures the break-even point: only after executing the workload EPT times does the cumulative energy saving from LQO compensate for the one-time training investment. A lower EPT indicates faster energy amortization and is preferable for deployment scenarios with limited workload repetition.

Boundary case. If $E_{\text{baseline}} < E_{\text{exec}}$, EROI is negative. If $0 < E_{\text{baseline}} - E_{\text{exec}} < E_{\text{infer}}$, execution saves energy but not enough to cover online optimization, so EPT is undefined. Experiments report these no-payback regimes separately from aggregate averages.

3.2 Framework Architecture

Figure 2 illustrates the overall architecture of E-LQO. The framework is designed around two parallel execution paths—the traditional query optimizer serving as a baseline, and the learned query optimization pipeline—unified by a comprehensive energy measurement subsystem. Unlike existing benchmarks that report only final latency improvements, E-LQO provides fine-grained, phase-level energy profiling across the entire LQO lifecycle, enabling practitioners to understand not merely *whether* LQO saves energy, but *where* energy is consumed and *when* the investment becomes worthwhile.

Traditional Query Optimizer (Baseline). The left branch of the framework executes queries using the database system's default optimizer (e.g., PostgreSQL's cost-based optimizer). This path establishes the energy baseline E_{baseline} against which LQO methods are compared. The energy measurement subsystem continuously monitors this execution to capture per-query baseline energy consumption.

Learned Query Optimization Pipeline. The right branch encompasses the four phases introduced in Equation 1. *Data Collection & Exploration* actively executes diverse plans to generate training samples (or, for passive methods, extracts features from existing logs); *Model Training* updates the learned components using the collected data; *Model Inference* applies the trained model to predict costs or select plans for incoming queries; and *Query Execution* physically executes the learned plans. Each phase operates with independent energy instrumentation, allowing the framework to attribute energy consumption to specific stages.

Energy Measurement & Profiling Subsystem. The foundation of E-LQO is a unified measurement layer that spans both execution paths. This subsystem employs fine-grained power sensors to capture energy consumption at the phase level, enabling the computation of metrics defined

in Section 3.1. Crucially, the subsystem tracks energy throughout the entire lifecycle rather than focusing solely on inference-time performance.

Outcome Analysis. The framework ultimately determines whether the LQO deployment results in *Outcome A: Energy Savings*, where the learned optimizer discovers better plans that reduce execution energy sufficiently to offset all overheads, or *Outcome B: Extra Consumption*, where the cumulative overhead of data collection, training, and inference exceeds the execution energy savings. By providing detailed phase-level breakdowns, E-LQO enables practitioners to diagnose the root causes of either outcome and identify optimization opportunities.

In summary, E-LQO distinguishes itself from prior evaluation approaches by offering lifecycle-wide energy transparency. Rather than delivering a binary verdict on energy efficiency, our framework quantifies the energy contribution of each phase, computes domain-specific metrics (EROI and EPT), and provides actionable insights for improving the energy efficiency of learned query optimization systems.

4 Methodology

This section details the energy measurement methodology employed by E-LQO across each phase of the LQO lifecycle. We describe the instrumentation techniques for data collection and exploration (§4.1), model training (§4.2), model inference (§4.3), and learned plan execution (§4.4). We then synthesize these measurements into lifecycle-wide energy metrics (§4.5) and discuss RAPL DRAM measurement uncertainty (§4.6).

4.1 Data Collection and Exploration Energy

The data collection phase constitutes the foundation of LQO training, wherein the system executes diverse query plans to gather training samples. To formalize energy measurement during this phase, we first extend the workload definition from Section 3 by partitioning the query set into training and test subsets.

Workload Partitioning. Let $Q = \{q_1, q_2, \ldots, q_n\}$ denote the complete workload. We partition Q into a training set $Q_{\text{train}} = \{q_1, q_2, \ldots, q_m\}$ and a test set $Q_{\text{test}} = \{q_{m+1}, \ldots, q_n\}$, where $m < n$. The data collection energy E_{collect} is measured exclusively over Q_{train}, as LQO methods utilize training queries to construct their experience pools and learn optimization strategies.

Baseline Energy Measurement. For each training query $q_i \in Q_{\text{train}}$, we first execute the plan selected by the traditional optimizer (PostgreSQL's default planner) and record the corresponding energy consumption $e^{(i)}_{\text{baseline}}$. The aggregate baseline energy over the training set is:

$$E^{\text{train}}_{\text{baseline}} = \sum_{i=1}^{m} e^{(i)}_{\text{baseline}} \tag{6}$$

This measurement establishes the energy reference against which LQO exploration overhead is quantified.

Exploration Energy Measurement. The exploration energy E_{collect} captures the additional energy[1] consumed by LQO methods when executing plans beyond the default optimizer's choices. This overhead varies substantially across different LQO approaches due to their distinct exploration strategies:

[1]SSD/device power is not reported as a separate energy rail. Device-level flash/SSD energy is difficult to isolate robustly because vendors expose only coarse power specifications and operation-level behavior depends on hidden device state and garbage collection [10, 19, 68]. In our setup, direct SSD-device power is small (about 3% of measured CPU+DRAM energy). Disk-I/O effects are captured through full query-window RAPL CPU-package and DRAM measurements: realistic-buffer runs in Section 5.3 lower buffer-hit ratios, lengthen execution/exploration windows, and change CPU/DRAM activity, all of which enter E_{baseline}, E_{collect}, and E_{exec}.

Per-method exploration energy. The three execution-based methods differ primarily in exploration breadth. **Bao** invokes the traditional optimizer under each of $|\mathcal{H}| \approx 25$ hint configurations, so $E^{\text{Bao}}_{\text{collect}} = \sum_{i=1}^{m} \sum_{h \in \mathcal{H}} e^{(i,h)}_{\text{exec}}$. **Balsa** runs T RL-training iterations that execute the generated plan for each q_i, yielding $E^{\text{Balsa}}_{\text{collect}} = \sum_{i=1}^{m} \sum_{t=1}^{T} e^{(i,t)}_{\text{exec}}$. **LEON** integrates with PostgreSQL's DP [42] and explores candidate plans for each equivalent set $S \in \mathcal{S}_i$, giving $E^{\text{LEON}}_{\text{collect}} = \sum_{i=1}^{m} \sum_{S \in \mathcal{S}_i} \sum_{p \in \text{Explore}(S)} e^{(p)}_{\text{exec}}$. Exploration-space cardinality: LEON > Balsa > Bao.

Energy Attribution. During the data collection phase, energy consumption is predominantly attributed to CPU and memory subsystems. Query plan execution involves intensive CPU computation for operator evaluation (e.g., hash computations, sorting, join processing) and substantial memory access for intermediate result materialization and buffer management. We employ hardware-level power monitoring through Intel's Running Average Power Limit (RAPL) interface [16, 18] to capture fine-grained energy measurements for CPU cores and DRAM modules throughout the exploration process. This full-window CPU+DRAM accounting captures I/O-induced stalls, buffer misses, and buffer-management work in the measured query window.

4.2 Model Training Energy

The model training phase encompasses the computational overhead of updating neural network parameters based on collected experiences. Unlike data collection, which is dominated by database execution, model training primarily consumes energy through gradient computation and weight updates on CPU and GPU resources.

Training Energy Components. The total training energy E_{train} comprises three hardware components:

$$E_{\text{train}} = E^{\text{CPU}}_{\text{train}} + E^{\text{MEM}}_{\text{train}} + E^{\text{GPU}}_{\text{train}} \tag{7}$$

where $E^{\text{CPU}}_{\text{train}}$ and $E^{\text{MEM}}_{\text{train}}$ capture the host-side computation for data preprocessing, batch construction, and gradient aggregation, while $E^{\text{GPU}}_{\text{train}}$ measures the accelerator energy for forward and backward passes through the neural network.

Handling Interleaved Exploration and Training. Reinforcement learning-based methods such as Balsa interleave plan exploration with model updates [40], executing queries and training the policy network within the same iteration loop. This interleaved paradigm presents a measurement challenge, as the two phases are temporally coupled. In E-LQO, we maintain phase separation by attributing energy to the appropriate lifecycle component based on the operation type:

- Energy consumed during query plan execution (regardless of whether it occurs within a training loop) is attributed to E_{collect}.
- Energy consumed during neural network forward/backward passes and parameter updates is attributed to E_{train}.

This decomposition is achieved by instrumenting the LQO codebase with phase-specific energy checkpoints, enabling precise attribution even when operations are temporally interleaved. Formally, for methods with interleaved execution, we decompose the joint energy measurement E_{joint} as:

$$E_{\text{joint}} = E_{\text{collect}} + E_{\text{train}} \tag{8}$$

where each component is measured by aggregating energy samples during the corresponding operation types.

Shared TreeCNN Backbone. TreeCNN [44] is the cost-model backbone adopted by all three execution-based optimizers in their original designs. Bao inherits the TreeCNN design from Neo [37], an end-to-end reinforcement-learning plan generator [59]; Balsa reuses the same TreeCNN topology in its bootstrap phase [69]; LEON employs a pairwise-ranking TreeCNN head in its bottom-up DP-integrated pipeline [5]. This shared design choice across the execution-based LQO family is

also documented in the comparative study of Lehmann et al. [27]. Each optimizer is evaluated under its own published architecture with hardware/software configurations held constant, and alternative cost-model architectures (QueryFormer [72], Zero-shot [14], DACE [33]) are evaluated in Section 5.5 to quantify the sensitivity of our lifecycle conclusions to this backbone choice.

4.3 Model Inference Energy

The model inference phase occurs during online query processing, where the trained model predicts costs or selects plans for incoming queries. Although Bao, Balsa, and LEON all employ TreeCNN-based models with comparable architectural complexity, their inference energy profiles differ substantially due to variations in the inference scope—the number and type of candidate plans evaluated per query.

Inference device and accounting scope. E-LQO supports both GPU-accelerated and CPU-only inference deployments. By default, learned-model forward passes execute on an NVIDIA Tesla T4 with energy measured via NVML; when forward passes are dispatched to the host CPU instead, they are profiled with the same RAPL instrumentation used elsewhere in the lifecycle. Candidate-plan generation and feature encoding always run on the CPU and are tracked via RAPL. Lifecycle metrics (EROI/EPT) are computed from the total E_{infer} defined in Eq. 2, which sums these three components; the per-method formulas below isolate the model-forward term e_{forward} because inference scope directly governs how often it is invoked.

Inference Scope Variations. The inference scope directly determines the number of forward passes through the cost model, and consequently, the inference energy overhead:

Per-method inference energy. Bao: $e_{\text{infer}}^{\text{Bao},(i)} = |\mathcal{H}| \cdot e_{\text{forward}}$ (evaluates all 25 hint configurations). **Balsa**: $e_{\text{infer}}^{\text{Balsa},(i)} = O(|R_i|) \cdot e_{\text{forward}} \cdot b$ where $|R_i|$ is relation count and b is beam width (beam search [8]). **LEON**: $e_{\text{infer}}^{\text{LEON},(i)} = \sum_{S \in \mathcal{S}_i} |P_S| \cdot e_{\text{forward}}$, invoking intra- and inter-equivalent-set models during DP enumeration; pruning via M_θ^O mitigates combinatorial growth.

Inference Energy Measurement. We measure online optimization energy by separately recording candidate-plan generation, feature encoding, and learned-model forward passes, then summing them into the total E_{infer} used in EROI/EPT. For each test query $q_i \in Q_{\text{test}}$, we record:

$$E_{\text{infer}} = \sum_{i=m+1}^{n} e_{\text{infer}}^{(i)} \tag{9}$$

where $e_{\text{infer}}^{(i)}$ is the per-query total online optimization overhead. This measurement employs GPU energy monitoring (via NVIDIA Management Library for GPU-based inference [49]) and CPU energy monitoring (via RAPL).

Planning and Candidate-Plan Generation Attribution. PostgreSQL planning energy is small (0.010–0.082 kJ per test-workload cycle, $< 0.1\%$ of E_{baseline}) and remains inside the baseline RAPL window. For execution-based LQO, we separately measure CPU-side candidate-plan generation (0.18–2.54 kJ across datasets) and GPU-side model forward, then sum them per Eq. 2; for passive/log-driven methods, plan generation and model evaluation are measured together in the end-to-end optimization path.

4.4 Learned Plan Execution Energy

The final phase of the LQO lifecycle is the execution of learned plans—the query plans selected by the trained model for test queries. This phase determines whether the energy invested in training yields runtime benefits.

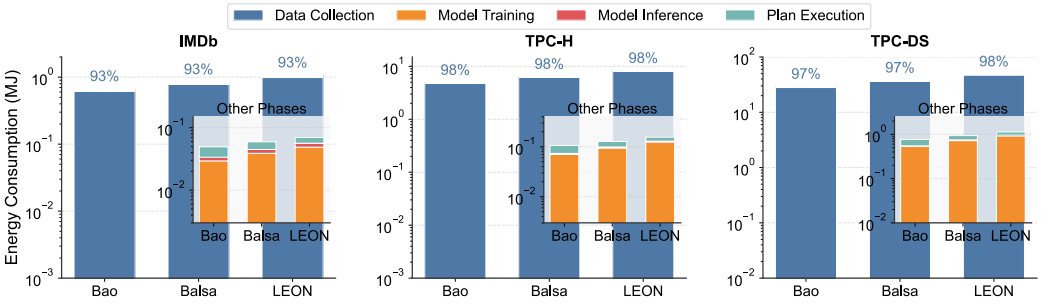


Fig. 3. Phase-level energy breakdown across the LQO lifecycle.

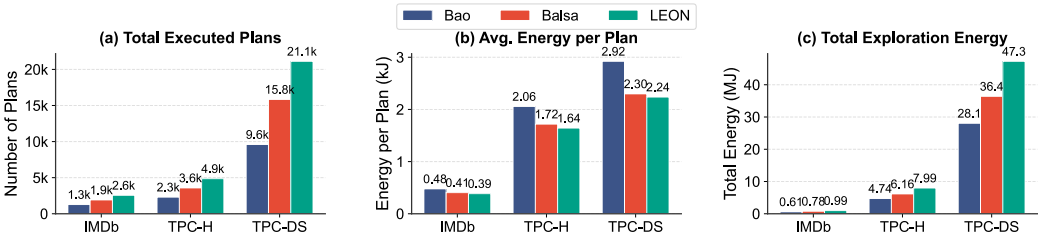


Fig. 4. Fine-grained decomposition of data collection energy.

Execution Energy Measurement. For each test query $q_i \in Q_{\text{test}}$, we execute the plan p_i^* selected by the LQO method and record the execution energy $e_{\text{exec}}^{(i)}$:

$$E_{\text{exec}} = \sum_{i=m+1}^{n} e_{\text{exec}}^{(i)} \tag{10}$$

The execution energy is measured using RAPL-based power monitoring throughout the query execution window, capturing CPU and memory energy consumption.

Baseline Comparison. To quantify the execution-phase benefits of LQO, we compare E_{exec} against the baseline execution energy on test queries:

$$E_{\text{baseline}}^{\text{test}} = \sum_{i=m+1}^{n} e_{\text{baseline}}^{(i)} \tag{11}$$

where $e_{\text{baseline}}^{(i)}$ is the execution energy when using the traditional optimizer's plan for query q_i. The difference $\Delta E_{\text{exec}} = E_{\text{baseline}}^{\text{test}} - E_{\text{exec}}$ represents the per-cycle energy savings attributable to improved plan quality.

Joint Latency and Energy Recording. E-LQO records both latency and energy metrics for each query execution; specifically, the per-query execution latency $\ell_{\text{exec}}^{(i)}$ is recorded alongside $e_{\text{exec}}^{(i)}$ for each test query, enabling correlation analysis between performance improvements and energy savings. This dual measurement reveals whether latency reductions translate proportionally to energy reductions, or whether certain plan optimizations (e.g., trading CPU for memory) exhibit different energy characteristics despite similar latency profiles.

Unified field convention. Across Section 5.2–5.6, E_{collect} / E_{prep} denotes active exploration or passive preparation, and E_{infer} follows Eq. 2.

4.5 Lifecycle Energy Synthesis and Metrics

Having established measurement methodologies for each phase, we now synthesize these measurements into actionable insights by analyzing phase interdependencies and characterizing the tradeoff landscape.

Phase Interdependencies. The four lifecycle phases exhibit fundamental tradeoffs rather than independent optimization targets. *Exploration-Accuracy Tradeoff*: Constraining the exploration space (as Bao does) reduces E_{collect} but degrades model accuracy, potentially increasing E_{exec}. Let α denote the exploration coverage ratio; the expected execution energy exhibits diminishing returns:

$$\mathbb{E}[E_{\text{exec}}(\alpha)] \approx E^*_{\text{exec}} + \frac{\beta}{1 + \gamma \cdot \alpha} \tag{12}$$

where E^*_{exec} is the optimal achievable execution energy, and $\beta, \gamma > 0$ are workload-dependent constants. This implies an energy-optimal exploration budget beyond which marginal gains diminish. *Online-Offline Tradeoff*: Methods like LEON incur higher per-query inference overhead but achieve superior plan quality, representing a tension between offline investment and online cost. The optimal balance depends on expected query volume: high-volume deployments favor lower inference overhead even at increased training cost.

Interpretation of β and γ. We interpret the two workload-dependent constants in Eq. 12 as follows: $\beta = E_{\text{exec}}(\alpha{=}0) - E^*_{\text{exec}}$ measures the per-cycle excess execution energy at zero template coverage, and γ controls the diminishing-returns rate; its half-saturation coverage $\alpha_{1/2} = 1/\gamma$ quantifies how quickly additional templates stop helping. This phenomenological fit is estimated from 18 per-subset measurement points per configuration and reported later in Section 5.5; across all six (dataset, method) combinations we find $\gamma \in [6, 16]$ ($\alpha_{1/2} \approx 6\%$–16.5%), showing that a small fraction of templates captures most of the attainable execution-energy reduction.

Metric Interrelationships. EROI and EPT capture complementary aspects of energy efficiency with a precise mathematical relationship:

$$\text{EPT} = \left\lceil \frac{E_{\text{collect}} + E_{\text{train}}}{(E^{\text{test}}_{\text{baseline}} - E_{\text{exec}}) \cdot (1 - \text{EROI}^{-1})} \right\rceil \tag{13}$$

For positive per-cycle net saving, $E^{\text{test}}_{\text{baseline}} - E_{\text{exec}} - E_{\text{infer}} > 0$, this expression is equivalent to Eq. 5; non-positive net saving leaves EPT undefined as described in Section 3.1. This reveals that high EROI does not guarantee low EPT—if absolute savings ($E^{\text{test}}_{\text{baseline}} - E_{\text{exec}}$) are small on already well-optimized workloads, even efficient inference cannot rapidly amortize training costs. Under this positive-net-saving condition, for deployment with expected query volume V, LQO is energy-beneficial if and only if $V > \text{EPT}$, defining a decision boundary where workloads with high repetition and substantial optimization potential favor LQO adoption.

Sensitivity to workload drift. EPT assumes stable workload characteristics, yet production environments exhibit distribution shift. Grounding in domain-adaptation theory [4, 35, 76], the target-domain error $\epsilon_T(h) \le \epsilon_S(h) + d_{\mathcal{H}\Delta\mathcal{H}}(S, T) + \lambda^*$; prior work on learned DB components [29, 30, 48] has confirmed such degradation under workload drift. Distribution shift affects EPT through two compounding mechanisms: per-cycle savings diminish as learned plans become suboptimal, and inference overhead yields diminishing returns as predictions lose accuracy. Let $\rho \in [0, 1]$ be the distributional similarity between training and deployment; following multiplicative error propagation,

$$\text{EPT}_{\text{eff}} \approx \text{EPT}/\rho^2, \tag{14}$$

with quadratic dependence because ρ affects both execution savings and amortisation rate. Section 5.5 validates this empirically.

4.6 RAPL DRAM Limitations and Confidence Intervals

DRAM energy is sampled from Intel RAPL's DRAM domain, for which prior work reports up to ±20% deviation under high-bandwidth memory traffic [18]. Because DRAM contributes only 8–14% of total measured energy in our TPC-DS workloads, this uncertainty translates to at most ±2.74% in total energy. This is smaller than the active–passive and scale-study gaps in Section 5.2–5.6, so our qualitative conclusions remain unchanged.

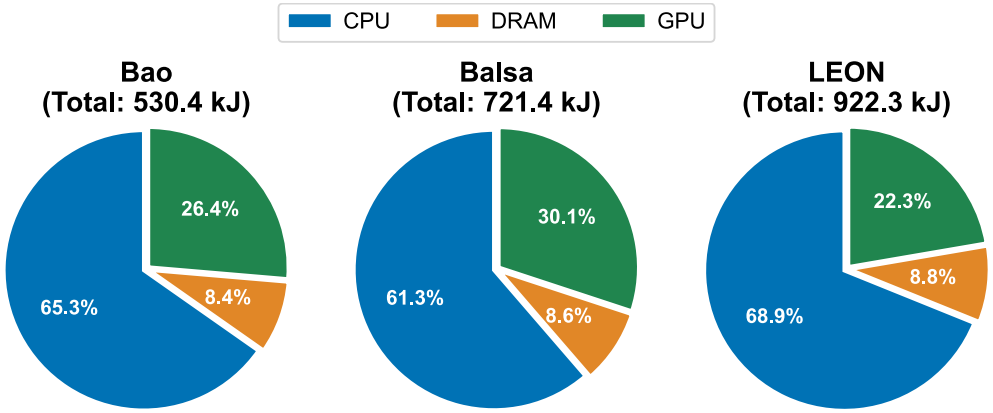


Fig. 5. Energy attribution during model training on TPC-DS 10 GB.

5 Experiment

5.1 Experimental Setup

Benchmarks and Workloads. We evaluate E-LQO on three widely-adopted benchmarks spanning diverse query complexities and data characteristics:

- **JOB (Join Order Benchmark)** [28]: Derived from the IMDb dataset, JOB comprises 113 queries across 33 templates with complex multi-way joins. We use 80 queries for training and 33 for testing (one per template).
- **TPC-H** [2]: From the 22 standard templates, we select 16 compatible with all LQO methods, excluding queries with nested subqueries unsupported by LEON and Balsa. Each template generates 10 instances with varying parameters, yielding 128 training and 32 test queries at 10 GB. For the scale study (Section 5.3), we additionally generate 30, 50, and 100 GB instances using `dbgen` to isolate the effect of larger data footprints and buffer pressure.
- **TPC-DS** [46]: We select 60 of 99 templates using the same compatibility criteria. Each template generates 10 instances, resulting in 480 training and 120 test queries at 10 GB. The same `dsgen`-based progressive scaling (30/50/100 GB) is used in Section 5.3, making TPC-H/TPC-DS the controlled scale-study workloads.

IMDb/JOB is substantially smaller (~3.6 GB) than the 10 GB TPC-H/TPC-DS instances and is kept at its standard scale to preserve JOB's commonly used semantics. Thus, IMDb reflects a light, memory-resident LQO deployment point, while TPC-H/TPC-DS provide the controlled 30/50/100 GB scale-study workloads. This distinction matters for EPT: smaller footprints shorten both useful execution savings and costly exploratory plans.

Hardware Configuration. All experiments are conducted on a server equipped with a 48-core Intel Xeon Platinum 8163 CPU (2.50GHz), 384GB of main memory, and a 2TB SSD. GPU-accelerated

training and inference utilize an NVIDIA Tesla T4. Energy measurements are collected via Intel RAPL for CPU and memory components, and NVIDIA Management Library (NVML) for GPU power consumption. Model inference hardware: NVIDIA Tesla T4 (GPU) by default; Intel Xeon Platinum 8163 (CPU) is used for the CPU-vs-GPU sensitivity study in Section 5.2.

Software Environment. We deploy PostgreSQL 17.5 [1, 56] as the underlying DBMS. The source code is publicly available.[2] Section 5.2 and the fixed-scale EROI/EPT summaries use a memory-resident 32 GB buffer configuration aligned with prior LQO evaluations. The progressive scale study in Section 5.3 uses realistic-buffer settings: 8/16/24/48 GB buffers for the 10/30/50/100 GB scale points, with the effective cache set to 3× the buffer size. PostgreSQL intra-query parallelism is enabled with 8 workers per gather. The RQ3 Parallel/Scaled protocol below is inter-query concurrency (multiple client connections), orthogonal to this operator-level PostgreSQL setting. All LQO methods (Bao, Balsa, LEON) follow their original specifications with TreeCNN as the shared cost-model backbone (Section 4.2).

Exploration Protocol. During data collection, we impose a timeout of 3× the default plan's latency for each query to prevent runaway exploratory plans. Plans exceeding this threshold are terminated and excluded from training. Reported executed-plan counts include only valid, de-duplicated exploratory executions that complete within the timeout, whereas the energy consumed by timed-out attempts is still charged to E_{collect} up to termination.

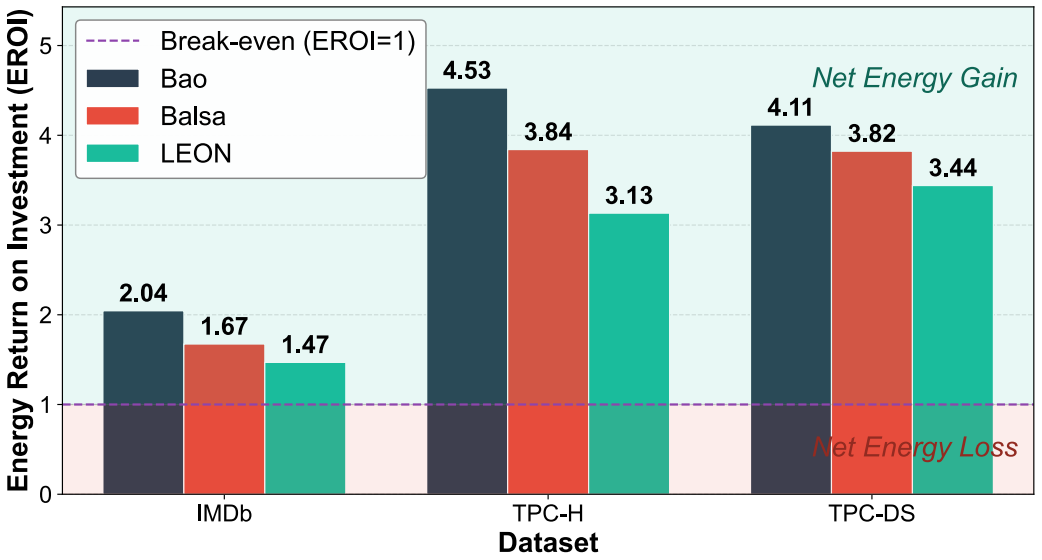


Fig. 6. EROI across methods and benchmarks under the fixed-scale memory-resident setting.

5.2 Energy Breakdown Analysis (RQ1)

To understand where energy is consumed throughout the LQO lifecycle, we conduct a comprehensive phase-level energy decomposition across all evaluated methods and benchmarks.

Lifecycle Energy Distribution. Figure 3 presents the energy consumption breakdown across the four lifecycle phases. The results reveal a striking pattern: for execution-based LQO, data collection

[2]https://github.com/liang-zibo/energy_lqo

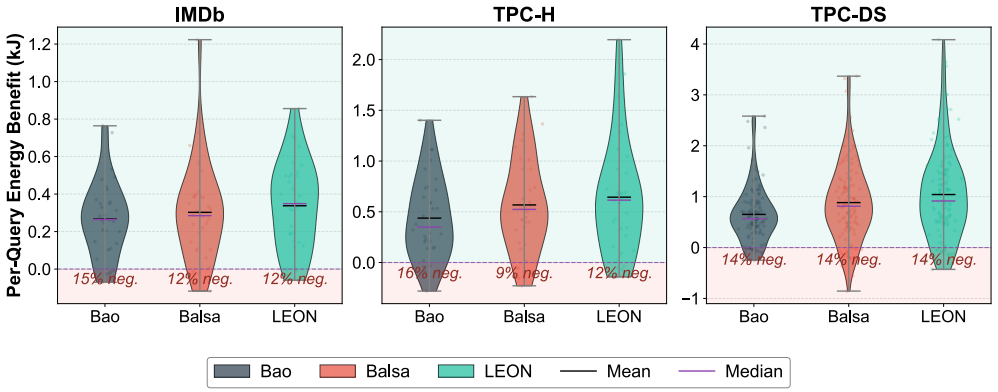


Fig. 7. Distribution of per-query energy benefits.

overwhelmingly dominates the total energy investment, accounting for 93% on IMDb, 98% on TPC-H, and 97–98% on TPC-DS across all three execution-based LQO methods. This disproportionate distribution stems from the fundamental requirement of executing numerous exploratory plans to generate sufficient training signals. In contrast, model training, model inference, and plan execution collectively contribute less than 7% of the total energy budget. Notably, the dominance of data collection energy is consistent across methods despite their vastly different exploration strategies—Bao's constrained hint space, Balsa's RL-based search, and LEON's white-box DP integration all exhibit similar phase-level distributions. Section 5.6 contrasts this execution-based pattern with passive/log-driven LQO.

Data Collection Phase Decomposition. Figure 4 provides a fine-grained analysis of the data collection phase by decomposing exploration energy into its constituent factors. As shown in Figure 4a, the total number of executed plans varies substantially across methods, with LEON exploring the most plans (2.6k, 4.9k, and 21.1k on IMDb, TPC-H, and TPC-DS, respectively) due to its integration with the optimizer's complete DP search space. Despite this larger exploration volume, Figure 4b reveals that per-plan energy decreases slightly from Bao to LEON (e.g., 2.92 kJ vs. 2.24 kJ on TPC-DS), attributable to LEON's pruning mechanisms. The net effect, in Figure 4c, is that total exploration energy scales proportionally with the number of explored plans.

Hardware Resource Attribution. Figure 5 shows training-phase energy across hardware components. CPU computation constitutes the dominant energy consumer (61.3–68.9% across methods), attributed to CPU-intensive plan tree encoding and CPU-GPU coordination overhead. GPU energy is second (22.3–30.1%); DRAM contributes only 8.4–8.8%.

Training micro-breakdown and inference memory. Table 1 reports TPC-DS training activity shares: CPU plan encoding (Enc, 40–48%) and GPU compute (22–30%) are the largest components, followed by GPU synchronization (Sync), CPU→GPU host-to-device transfer (H2D), and DRAM/other host activity (DRAM+). The entries are normalized activity shares rather than a mutually exclusive energy decomposition, so rows need not sum to 100%. Per-query GPU memory (Mem p95) remains modest (119–480 MB, far below the T4's 16 GB), so online inference is bounded by repeated model-forward invocations rather than capacity.

CPU vs. GPU model-forward energy. Table 2 contrasts model-forward inference on TPC-DS between the default Tesla T4 GPU and the Xeon Platinum 8163 CPU deployment. Switching from GPU to CPU increases per-query forward-pass energy by 22–44% across Bao/Balsa/LEON because runtime grows by 77–98%; LEON suffers most due to its dense pairwise scoring inside

Table 1. TPC-DS training/inference micro-breakdown. Enc/H2D/GPU/Sync/DRAM+ are non-exclusive normalized activity shares; Mem p95 is the 95th-percentile per-query GPU memory footprint.

Method	Enc	H2D	GPU	Sync	DRAM+	Mem p95
Bao	40.6	6.4	26.5	14.6	8.3	119 MB
Balsa	44.1	7.8	30.1	13.1	8.5	302 MB
LEON	47.8	8.7	22.3	11.2	10.0	480 MB

Table 2. CPU-vs-GPU model-forward inference on TPC-DS. CPU forward-pass energy is 20–44% higher because runtime grows by 76–98%.

	Bao		Balsa		LEON	
	kJ	s	kJ	s	kJ	s
GPU (T4)	17.62	229	25.78	336	33.59	437
CPU (Xeon)	21.50	405	33.40	615	48.30	865
Δ	+22%	+77%	+30%	+83%	+44%	+98%

Table 3. EPT across methods under the fixed-scale memory-resident setting. Values use unified $E^{\text{total}}_{\text{infer}}$ accounting; lower is better.

Dataset	Bao	Balsa	LEON
IMDb	**144**	204	291
TPC-H	**443**	468	577
TPC-DS	483	**476**	547

DP enumeration. The Tesla T4 numbers are used elsewhere in the paper as the default inference configuration, but CPU-only deployments retain a positive EROI on long-running workloads at the cost of longer payback. The complementary backbone sensitivity (TreeCNN/QueryFormer/Zero-shot/DACE) is reported later in Section 5.5.

5.3 Energy Break-even Analysis (RQ2)

We now investigate when data-collection investment yields net energy benefits during deployment. Two complementary metrics: EROI, the per-cycle efficiency of inference, and EPT, the amortization horizon.

Inference-Phase Energy Efficiency. Figure 6 presents EROI across all method–benchmark combinations. All configurations achieve EROI > 1, i.e., per-cycle savings exceed inference overhead. Bao consistently leads (lightweight 25-hint inference); LEON's deeper DP integration incurs higher per-query overhead.

Training Cost Amortization. Table 3 quantifies EPT under the unified $E^{\text{total}}_{\text{infer}}$ accounting: break-even ranges from 144 (Bao on IMDb) to 577 (LEON on TPC-H). EPT is shaped by benchmark complexity (TPC-H/DS 443–577 vs. IMDb 144–291), exploration breadth, and per-cycle net saving; larger savings can offset higher exploration cost, as Balsa slightly outpays Bao on TPC-DS.

Query-Level Benefit Distribution. Figure 7 shows that 9–16% of queries have negative energy benefit. Bao's distribution is tighter; LEON/Balsa have wider spread.

Table 4. Per-class net saving (%) by baseline-latency quantiles (S: lower 40%; M: middle 40%; L: upper 20%). Negative values mean LQO uses more energy than baseline.

	Bao			Balsa			LEON		
	S	M	L	S	M	L	S	M	L
IMDb	−21	+6	+28	−26	+3	+28	−33	−4	+27
TPC-H	−11	+19	+29	−17	+22	+38	−22	+24	+40
TPC-DS	−9	+13	+27	−14	+17	+35	−21	+19	+41

Table 5. Progressive scale study (realistic buffer). Buf. is `shared_buffers` in GB. Cells report EROI/EPT; E_{base} is in kJ.

DB	GB	Buf.	Hit	E_{base}	Bao	Balsa	LEON
TPC-H	10	8	0.98	62	4.4/467	4.1/442	3.5/509
	30	16	0.73	244	18.2/279	14.4/307	12.2/349
	50	24	0.60	509	37.3/274	31.9/273	24.9/320
	100	48	0.47	1253	79.0/280	67.5/292	53.0/326
TPC-DS	10	8	0.95	359	4.6/437	4.0/445	3.5/540
	30	16	0.69	1603	19.2/325	17.4/337	16.1/366
	50	24	0.54	3392	37.0/351	33.7/356	28.7/422
	100	48	0.43	8615	78.0/376	76.0/364	65.9/420

Query-stratified net saving. Table 4 stratifies per-class net saving by baseline-latency quantile. The Long class is uniformly positive, whereas the Short class is uniformly negative because the fixed $E_{\text{infer}}^{\text{total}}$ exceeds the achievable execution savings on short queries; the Middle class lies in between and is workload-dependent. LQO is therefore clearly more valuable on long-running queries, which alone reliably amortize the per-query inference overhead.

Progressive scale study. Under realistic buffers, lower hit ratios make TPC-H/TPC-DS more I/O bound and raise the per-cycle value of improved plans (Table 5). Yet active exploration also becomes costlier, so EPT remains in the hundreds even as EROI rises with scale.

5.4 LQO vs. Hardware Scaling (RQ3)

We compare PostgreSQL, Bao, Balsa, and LEON under the same inter-query concurrency protocol: Serial (1 conn), Parallel (6 conn, 1 socket), and Scaled (12 conn, 2 sockets). Table 6 reports total energy and latency.

Concurrency effects. Under inter-query concurrency, LQO's per-query optimization contends for the same resources: candidate enumeration and plan-tree encoding occupy the CPU sockets serving execution, and model-forward inference is serialized through the single shared T4. With 6–12 connections these stages queue both against each other and against query workers, inflating per-query optimization latency. The effect is workload-dependent: on IMDb (short memory-resident queries), this overhead exceeds the plan-quality benefit, flipping both metrics against PostgreSQL (+3–14% energy and +19–56% latency across Bao–LEON in Parallel/Scaled). On TPC-H/TPC-DS, longer per-query execution dilutes the overhead and LQO improves both metrics simultaneously: up to 17% energy and 8% latency saving across (method, mode) cells.

Table 6. Symmetric concurrency evaluation: total online energy (kJ) / total latency (s) under Serial (1 conn), Parallel (6 conn, 1 socket), and Scaled (12 conn, 2 sockets).

Dataset	Method	Serial	Parallel	Scaled
IMDb	PG	24.5/263.0	**11.3/75.3**	**12.6/45.4**
	Bao	**20.0/226.7**	11.6/89.6	13.8/69.2
	Balsa	20.5/234.4	12.9/93.5	13.2/64.0
	LEON	20.9/241.4	12.7/95.6	14.0/70.8
TPC-H	PG	46.4/491.3	20.9/132.8	23.5/78.7
	Bao	35.6/390.9	18.9/129.6	21.4/76.2
	Balsa	33.0/364.7	**17.3/122.3**	**19.5/74.5**
	LEON	**32.3/358.4**	18.8/125.7	22.8/79.1
TPC-DS	PG	292/3098	131/815	147/483
	Bao	233/2471	**119/776**	**134/451**
	Balsa	214/2290	125/783	141/469
	LEON	**204/2198**	127/799	137/459

The energy–latency co-improvement on TPC-H/TPC-DS reflects a double dividend: shorter wall-clock and lower time-integrated socket power. PG sustains ~157–161 W (Parallel) and ~299–304 W (Scaled), close to single/dual-socket TDP; the best LQO reaches ~142–153 W and ~262–297 W because T4 serialization leaves CPU sockets partially idle. LEON has the deepest Serial gains (−30%/−27% on TPC-H and −30%/−29% on TPC-DS), but its dense pairwise scoring drives the heaviest model-forward workload and competes most under concurrency, so the lead shifts: Balsa wins TPC-H Parallel/Scaled with lighter MCTS expansion, Bao's 25-hint path leads TPC-DS Parallel/Scaled, and LEON's margin shrinks to the smallest in those modes.

5.5 Sensitivity Analysis

Impact of Cost Estimation Models. Table 7 compares cost-model backbones (TreeCNN/QueryFormer/Zero-shot/DACE) for Bao on TPC-DS 10 GB under unified $E_{\text{infer}}^{\text{total}}$ accounting. Because this sensitivity keeps Bao's active exploration protocol fixed, E_{off} always includes Bao's TPC-DS data-collection energy. DACE attains the lowest EPT (312) and highest EROI (6.18), even though TreeCNN saves slightly more latency (20.2% vs. 19.7%): its lightweight architecture simultaneously cuts E_{exec} (182.8 vs. 210–214 kJ) and E_{infer} (17.7 vs. up to 25.2 kJ). Heavier backbones (QueryFormer, Zero-shot) raise inference cost beyond their marginal execution gains, pushing payback to 505–527 cycles. Section 5.6 evaluates DACE again as a passive/log-driven optimizer under incremental-cost accounting. Model complexity therefore does not monotonically improve energy efficiency—the latency-to-energy ratio matters more than latency alone.

Impact of Training Data Volume and Distribution Shift. Figure 8 shows that EPT and EROI both rise monotonically with the training-data fraction [64], but with markedly different slope profiles. EPT grows near-linearly across 10%–100% because offline debt scales with the training fraction, and the slope is only mildly attenuated by improving plan quality. EROI, in contrast, rises steeply at low fractions and saturates above ~70% as additional exploration yields rapidly diminishing per-cycle execution savings. Across this range, Bao is the most stable while LEON's EPT varies by more than 400 cycles between the 10% and 80% configurations. The 100% endpoint matches Table 3 under the unified $E_{\text{infer}}^{\text{total}}$ accounting. This experiment varies the fraction of training

Table 7. Bao backbone sensitivity on TPC-DS 10 GB memory-resident. E_{off} is in MJ; E_{exec} and E_{infer} are in kJ; L% is latency saving vs. PostgreSQL.

Backbone	E_{off}	E_{exec}	E_{infer}	L%	EROI	EPT
TreeCNN	28.6	214.0	19.04	20.2	4.11	483
QueryFormer	29.5	211.5	22.31	19.6	3.62	505
Zero-shot	30.0	210.1	25.20	18.9	3.26	527
DACE	28.6	182.8	17.72	19.7	6.18	312

instances under a fixed template set; the template-coverage experiment below separately studies template-count sensitivity.

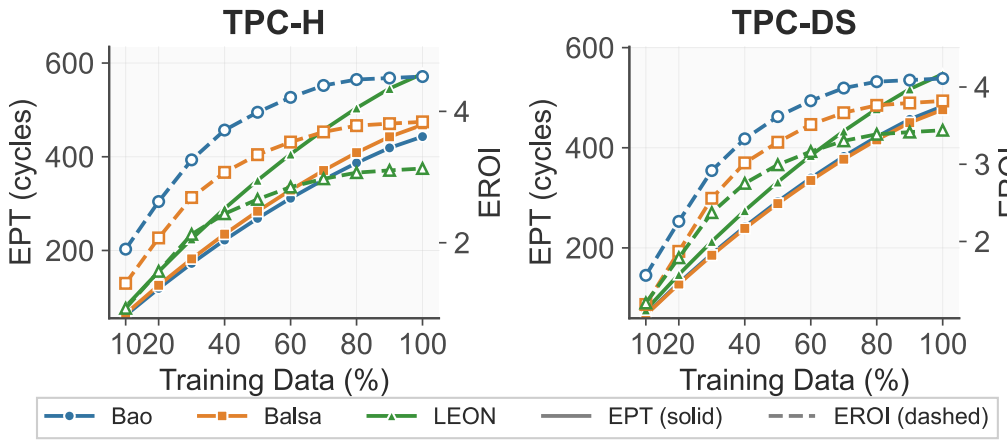


Fig. 8. Impact of training data volume on EPT and EROI.

Under controlled distribution shift (random inserts/deletes), Figure 9 shows that deletion yields lower EPT than equal-magnitude insertion; EPT rises super-linearly, consistent with the ρ^{-2} dependence in Eq. 14. The measured curves usually remain below the conservative bound, with improvement varying by method and shift magnitude (0–62% across the plotted points, averaging about 26–30%), so the bound should be read as an upper envelope rather than a fixed gap.

Template-Count Sensitivity. We fit template-coverage sensitivity using six coverage levels on TPC-H/TPC-DS with three stratified subsets per point. Figure 10 plots the EPT/EROI surface across the swept α values and Table 8 reports four representative TPC-DS operating points. With fixed E_{infer} per method/dataset, the key pattern is diminishing returns: very low coverage under-trains the model, while coverage beyond $\alpha \approx 0.5$ increases E_{collect} faster than it lowers E_{exec}. The shaded band in Figure 10 ($\alpha^* \in [0.46, 0.57]$) marks the payback-optimal region empirically observed across all three execution-based methods.

Template-Coverage Fit. Fitting Eq. 12 on 18 stratified points per configuration gives $R^2 \in [0.986, 0.993]$ with residuals matching RAPL noise. We define α_{undef} as the coverage at which per-cycle net saving vanishes, i.e., $E^{\text{test}}_{\text{baseline}} - E_{\text{exec}}(\alpha_{\text{undef}}) - E_{\text{infer}} = 0$; substituting Eq. 12 yields the closed form $\alpha_{\text{undef}} = \frac{1}{\gamma}\left(\frac{\beta}{E^{\text{test}}_{\text{baseline}} - E_{\text{infer}} - E^*_{\text{exec}}} - 1\right)$, so any $\alpha < \alpha_{\text{undef}}$ leaves EPT undefined per the boundary case in Section 3.1. Table 9 reports the resulting deployment boundaries: $\alpha^* \in [0.46, 0.57]$

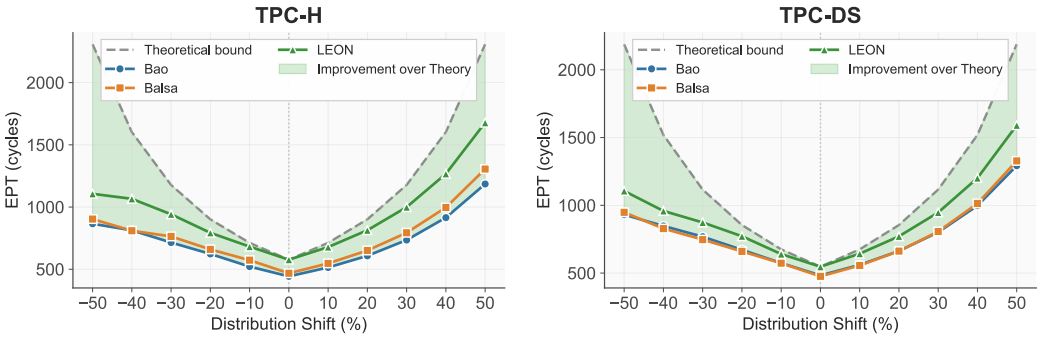


Fig. 9. EPT sensitivity to workload distribution shift. Negative shift denotes deletions; positive shift denotes insertions.

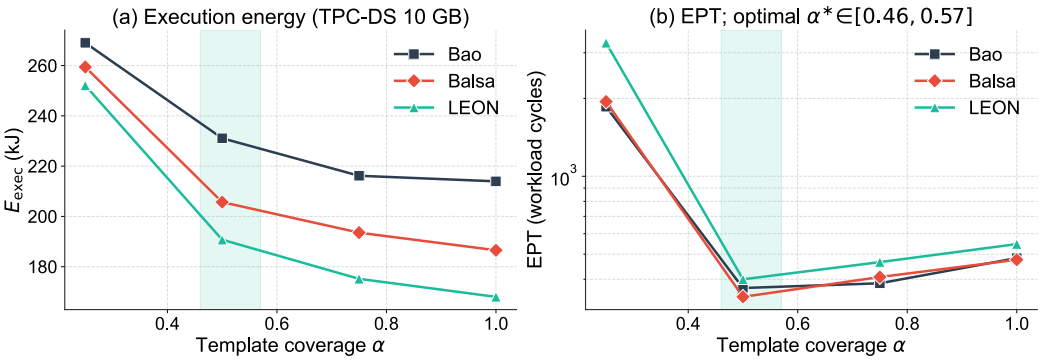


Fig. 10. Template-coverage sensitivity on TPC-DS 10 GB; shaded band marks $\alpha^* \in [0.46, 0.57]$.

Table 8. TPC-DS template sensitivity: E_{collect} (MJ), E_{exec} (kJ), and EPT.

	Bao			Balsa			LEON		
α	E_c	E_e	EPT	E_c	E_e	EPT	E_c	E_e	EPT
0.25	7.6	269	1857	9.8	259	1944	12.9	252	3278
0.50	15.3	231	**370**	19.7	206	**342**	25.6	191	**400**
0.75	21.6	216	386	28.3	194	408	37.0	175	466
1.00	28.1	214	483	36.4	187	476	47.3	168	547

is payback-optimal, while $\alpha_{\text{undef}} \in [0.201, 0.255]$ marks where online overhead exhausts per-cycle savings.

5.6 Passive / Log-driven Methods: Lifecycle Evaluation

Bao, Balsa, and LEON are execution-based; passive/log-driven LQO reuses supervision such as logs, labels, samples, or plan-latency records [52]. We evaluate NeuroCard [70], MSCN [20], Zero-shot [14], and DACE [33] under incremental-cost accounting: E_{prep} charges the additional feature extraction or sample scans needed to consume available supervision, and EPT reflects deployment

Table 9. Fitted parameters for $E_{\text{exec}}(\alpha) = E^*_{\text{exec}} + \beta/(1+\gamma\alpha)$. E^*_{exec} and β are in kJ.

Dataset	Method	E^*_{exec}	β	γ	R^2	α^*	α_{undef}
TPC-H	Bao	29.58	40.60	9.73	0.9892	0.46	0.201
	Balsa	21.63	46.91	6.08	0.9932	0.54	0.220
	LEON	18.65	65.96	8.99	0.9923	0.55	0.236
TPC-DS	Bao	184.1	330.9	10.64	0.9920	0.57	0.255
	Balsa	148.6	444.7	12.92	0.9914	0.49	0.219
	LEON	137.2	535.2	15.68	0.9862	0.49	0.223

with reusable logs or labels. Figure 11 contrasts the lifecycle composition of active and passive methods at TPC-DS 100 GB under this accounting, highlighting that exploration debt no longer dominates once supervision can be reused.

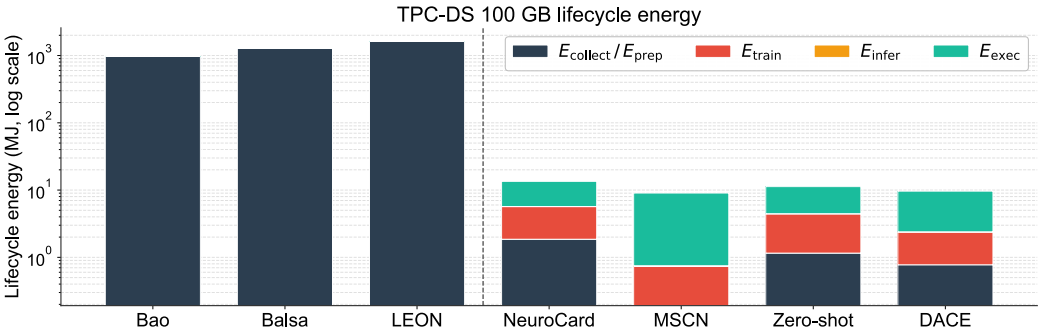


Fig. 11. Lifecycle energy at TPC-DS 100 GB: active vs passive methods under incremental-cost accounting.

Table 10. Passive/log-driven metrics at 10 GB memory-resident scale. E_{off} and E_{on} are in kJ.

DB	Method	E_{off}	E_{on}	EROI	EPT
IMDb	NeuroCard	265.3	23.76	1.37	374
	MSCN	150.6	23.80	1.50	225
	Zero-shot	1712.6	23.33	1.23	1503
	DACE	572.3	22.10	1.77	242
TPC-H	NeuroCard	522.6	42.41	2.38	131
	MSCN	195.7	42.55	2.84	51
	Zero-shot	1858.0	37.57	2.98	211
	DACE	747.9	37.89	3.80	88
TPC-DS	NeuroCard	1330.7	249.15	4.26	31
	MSCN	369.4	260.07	4.11	12
	Zero-shot	2604.1	217.84	4.44	35
	DACE	1163.6	220.81	5.71	17

10 GB memory-resident configuration. Passive payback varies widely: Table 10 ranges from 225–1503 cycles on IMDb, 51–211 on TPC-H, and 12–35 on TPC-DS. Heavy passive models are less suitable for light workloads: MSCN usually has the lowest EPT, followed by DACE, while NeuroCard and Zero-shot vary by workload, with Zero-shot particularly costly on IMDb/TPC-H. This ranking differs from the latency-save ranking (Zero-shot $\gtrsim$ DACE > NeuroCard > MSCN).

Scale study under realistic buffer. We re-run passive methods on TPC-H/TPC-DS at 10–100 GB with `shared_buffers` $\in \{8, 16, 24, 48\}$ GB for the 10/30/50/100 GB scale points. Table 10 reports the separate 10 GB memory-resident passive setting, whereas Fig. 12 and Table 11 use the realistic-buffer scale-study setting throughout. TPC-H follows the same trend as the TPC-DS summary in Fig. 12 and Table 11. At 100 GB, passive EPT falls to 2–6 cycles while active EPT remains 364–420, separating execution-based exploration debt from passive preparation cost.

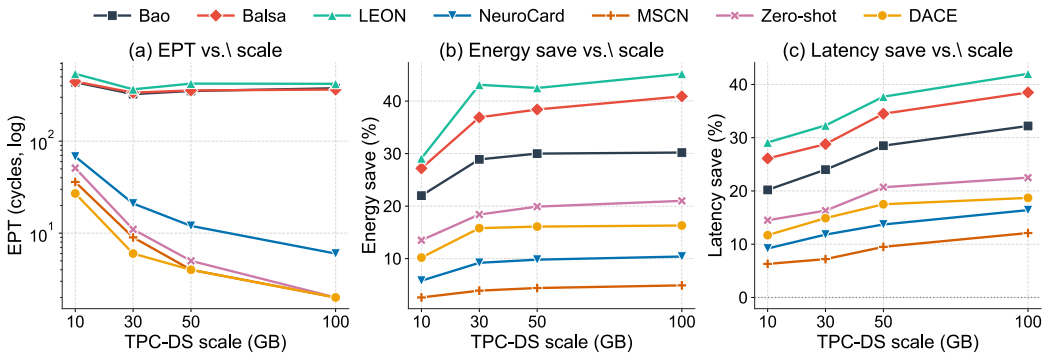


Fig. 12. Passive vs active at TPC-DS scale: EPT (log), energy-save, and latency-save.

Table 11. Active vs. passive on TPC-DS realistic-buffer settings. E_t is offline energy (MJ); E% is net online energy saving vs PG including total inference overhead; L% is latency saving vs PG.

Method	TPC-DS 10 GB				TPC-DS 100 GB			
	E_t	E%	L%	EPT	E_t	E%	L%	EPT
Bao	34.5	22.0	20.2	437	977.9	30.2	32.2	376
Balsa	43.5	27.2	26.1	445	1278.9	40.9	38.5	364
LEON	56.4	29.1	29.1	540	1635.5	45.2	42.0	420
NeuroCard	1.42	5.8	9.2	68	5.68	10.4	16.4	6
MSCN	0.34	2.6	6.3	36	0.74	4.9	12.1	**2**
Zero-shot	2.46	13.5	14.5	51	4.44	21.0	22.5	**2**
DACE	1.00	10.2	11.7	**27**	2.38	16.3	18.7	**2**

Interpretation of Table 11. Under realistic buffers, active methods give stronger latency gains at 10 GB (20–29%) and 100 GB (32–42%) but require hundreds-cycle payback. Passive methods amortize an order of magnitude faster under existing-log accounting (27–68 cycles at 10 GB; 2–6 at 100 GB) yet trail active latency at both scales: passive L% is 6.3–14.5% vs. 20.2–29.1% (10 GB) and 12.1–22.5% vs. 32.2–42.0% (100 GB), a 15–30 pp gap. Within passive methods, plan-level Zero-shot/DACE retain the strongest latency saving and lowest EPT (DACE leads at 10 GB; MSCN/Zero-shot/DACE tie at

2 cycles at 100 GB), while MSCN's tiny training cost keeps its EPT competitive despite the smallest latency gain. All numbers are averaged over three repeats with total inference energy (including planning), and RAPL DRAM uncertainty changes total energy by only ±1.65–2.74%.

6 Conclusion

We presented E-LQO, a lifecycle energy-evaluation framework spanning data collection/preparation, training, total online inference, and plan execution. Across seven optimizers, LQO's energy value is conditional: execution-based methods deliver better plans but carry hard-to-amortize exploration debt, while passive/log-driven methods amortize quickly with reusable supervision yet may underperform on light workloads. Practical deployment should stop exploration near diminishing returns, reuse passive supervision when available, and favor PostgreSQL with hardware parallelism for short or highly concurrent workloads.

Future work should reduce plan-encoding cost, lower the cost of obtaining training logs, and cut the fixed per-query inference overhead and its contention with concurrent workers, so LQO's advantage holds under inter-query concurrency.

Acknowledgments

This work is partially supported by NSFC (No. 62472068).

References

[1] [n. d.]. PostgreSQL. https://www.postgresql.org
[2] Melyssa Barata, Jorge Bernardino, and Pedro Furtado. 2015. An overview of decision support benchmarks: TPC-DS, TPC-H and SSB. *New Contributions in Information Systems and Technologies: Volume 1* (2015), 619–628.
[3] Luiz André Barroso and Urs Hölzle. 2007. The case for energy-proportional computing. *Computer* 40, 12 (2007), 33–37.
[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. *Machine learning* 79, 1 (2010), 151–175.
[5] Xu Chen, Haitian Chen, Zibo Liang, Shuncheng Liu, Jinghong Wang, Kai Zeng, Han Su, and Kai Zheng. 2023. Leon: A new framework for ml-aided query optimization. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2261–2273.
[6] Sudipto Das, Feng Li, Vivek R Narasayya, and Arnd Christian König. 2016. Automated demand-driven resource scaling in relational database-as-a-service. In *Proceedings of the 2016 International Conference on Management of Data*. 1923–1934.
[7] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. 2015. Data center energy consumption modeling: A survey. *IEEE Communications surveys & tutorials* 18, 1 (2015), 732–794.
[8] Markus Freitag and Yaser Al-Onaizan. 2017. Beam Search Strategies for Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*. 56–60.
[9] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. 2014. Deep autoregressive networks. In *International Conference on Machine Learning*. PMLR, 1242–1250.
[10] Laura M Grupp, Adrian M Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H Siegel, and Jack K Wolf. 2009. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, New York, NY, USA, 24–33.
[11] Binglei Guo, Jiong Yu, Dexian Yang, Hongyong Leng, and Bin Liao. 2022. Energy-efficient database systems: A systematic survey. *Comput. Surveys* 55, 6 (2022), 1–53.
[12] Stavros Harizopoulos, Mehul Shah, Justin Meza, and Parthasarathy Ranganathan. 2009. Energy Efficiency: The New Holy Grail of Data Management Systems Research. In *CIDR*.
[13] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How good are learned cost models, really? Insights from query optimization tasks. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–27.
[14] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proc. VLDB Endow.* 15, 11 (July 2022), 2361–2374. doi:10.14778/3551793.3551799
[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[16] Intel. 2024. Running Average Power Limit Energy Reporting. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html

[17] Yannis Ioannidis. 2003. The History of Histograms (Abridged). In *Proceedings 2003 VLDB Conference*. Elsevier, 19–30.
[18] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3, 2 (2018), 1–26.
[19] Jaeho Kim, Donghee Lee, and Sam H Noh. 2015. Towards SLO Complying SSDs Through OPS Isolation. In *13th USENIX Conference on File and Storage Technologies*. USENIX Association, Santa Clara, CA, USA, 183–189.
[20] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned cardinalities: Estimating correlated joins with deep learning. In *Conference on Innovative Data Systems Research (CIDR)*.
[21] Thomas Kissinger, Dirk Habich, and Wolfgang Lehner. 2018. Adaptive energy-control for in-memory database systems. In *Proceedings of the 2018 International Conference on Management of Data*. 351–364.
[22] Bartłomiej Kocot, Paweł Czarnul, and Jerzy Proficz. 2023. Energy-aware scheduling for high-performance computing systems: A survey. *Energies* 16, 2 (2023), 890.
[23] Mustafa Korkmaz, Martin Karsten, Kenneth Salem, and Semih Salihoglu. 2018. Workload-aware CPU performance scaling for transactional database systems. In *Proceedings of the 2018 International Conference on Management of Data*. 291–306.
[24] Willis Lang, Stavros Harizopoulos, Jignesh M Patel, Mehul A Shah, and Dimitris Tsirogiannis. 2012. Towards Energy-Efficient Database Cluster Design. *Proceedings of the VLDB Endowment* 5, 11 (2012).
[25] Willis Lang and Jignesh Patel. 2009. Towards Eco-friendly Database Management Systems. In *CIDR*.
[26] Etienne Le Sueur and Gernot Heiser. 2010. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*. 1–8.
[27] Claude Lehmann, Pavel Sulimov, and Kurt Stockinger. 2024. Is Your Learned Query Optimizer Behaving As You Expect? A Machine Learning Perspective. *Proceedings of the VLDB Endowment* 17, 7 (2024), 1565–1577.
[28] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.
[29] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently adapting learned cardinality estimators to data and workload drifts. In *Proceedings of the 2022 International Conference on Management of Data*. 1920–1933.
[30] Sijia Li, Peng Cai, Yiqi Shen, Huiqi Hu, Rong Zhang, Xuan Zhou, Xuquan Qing, and Ri Zhao. 2024. SPQO: Learning to Safely Reuse Cached Plans for Dynamic Workloads. In *Database Systems for Advanced Applications (Lecture Notes in Computer Science, Vol. 14850)*. Springer Nature Singapore, 315–330. doi:10.1007/978-981-97-5552-3_21
[31] Sijia Li, Peng Cai, Zhifan Zhang, Huiqi Hu, Rong Zhang, Xuan Zhou, Quanqing Xu, and Chuanhui Yang. 2025. APQO: An Adaptive Framework for Parametric Query Optimization. *Proceedings of the ACM on Management of Data* 3, 6, Article 296 (2025), 25 pages. doi:10.1145/3769761
[32] Wei Li, Jiachi Zhang, Ye Yin, Yan Li, Zhanyang Zhu, Wenchao Zhou, Liang Lin, and Feifei Li. 2024. Flux: decoupled auto-scaling for heterogeneous query workload in Alibaba AnalyticDB. In *Companion of the 2024 international conference on management of data*. 255–268.
[33] Zibo Liang, Xu Chen, Yuyang Xia, Runfan Ye, Haitian Chen, Jiandong Xie, and Kai Zheng. 2024. DACE: A Database-Agnostic Cost Estimator. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4925–4937.
[34] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J Gordon. 2018. Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*. 631–645.
[35] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. 2009. Domain Adaptation: Learning Bounds and Algorithms. (2009).
[36] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*. 1275–1288.
[37] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1705–1718.
[38] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1733–1746. doi:10.14778/3342263.3342646
[39] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. 2020. Recalibrating global data center energy-use estimates. *Science* 367, 6481 (2020), 984–986.
[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[41] Songsong Mo, Yue Zhao, Zhifeng Bao, Quanqing Xu, Chuanhui Yang, and Gao Cong. 2024. RankPQO: Learning-to-Rank for Parametric Query Optimization. *Proceedings of the VLDB Endowment* 18, 3 (2024), 863–875. doi:10.14778/3712221.3712248

[42] Guido Moerkotte and Thomas Neumann. 2006. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 930–941.

[43] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment* 2, 1 (2009), 982–993.

[44] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[45] David J Murphy and Charles AS Hall. 2011. Energy return on investment, peak oil, and the end of economic growth. *Annals of the New York Academy of Sciences* 1219, 1 (2011), 52–72.

[46] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 1049–1058.

[47] Vikram Nathan, Vikramank Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Intelligent Scaling in Amazon Redshift. In *Companion of the 2024 International Conference on Management of Data*. 269–279.

[48] Parimarjan Negi, Ziniu Wu, Andreas Kipf, Nesime Tatbul, Ryan Marcus, Sam Madden, Tim Kraska, and Mohammad Alizadeh. 2023. Robust query driven cardinality estimation under changing workloads. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1520–1533.

[49] NVIDIA. 2024. NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml.

[50] Iraklis Psaroudakis, Thomas Kissinger, Danica Porobic, Thomas Ilsche, Erietta Liarou, Pınar Tözün, Anastasia Ailamaki, and Wolfgang Lehner. 2014. Dynamic fine-grained scheduling for energy-efficient main-memory queries. In *Proceedings of the Tenth International Workshop on Data Management on New Hardware*. 1–7.

[51] Iraklis Psaroudakis, Tobias Scheuer, Norman May, and Anastasia Ailamaki. 2013. Task scheduling for highly concurrent analytical and transactional main-memory workloads. In *Proceedings of the Fourth International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS 2013)*.

[52] Silvan Reiner and Michael Grossniklaus. 2023. Sample-Efficient Cardinality Estimation Using Geometric Deep Learning. *Proceedings of the VLDB Endowment* 17, 4 (2023), 740–752.

[53] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*. 23–34.

[54] Greg Semeraro, Grigorios Magklis, Rajeev Balasubramonian, David H Albonesi, Sandhya Dwarkadas, and Michael L Scott. 2002. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*. IEEE, 29–40.

[55] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1348–1362.

[56] Michael Stonebraker and Lawrence A Rowe. 1986. The Design of POSTGRES. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*. 340–355.

[57] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 3645–3650.

[58] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proceedings of the VLDB Endowment* 13, 3 (2019), 307–319. doi:10.14778/3368289.3368296

[59] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[60] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*. 1556–1566.

[61] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A Shah. 2010. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 231–242.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[63] Benjamin Wagner, André Kohn, and Thomas Neumann. 2021. Self-tuning query scheduling for analytical workloads. In *Proceedings of the 2021 international conference on management of data*. 1879–1891.

[64] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3 (2016), 1–40.

[65] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query execution time prediction in amazon redshift. In *Companion of the 2024 International Conference on Management of Data*. 280–294.

[66] Ziniu Wu, Markos Markakis, Chunwei Liu, Peter Baile Chen, Balakrishnan Narayanaswamy, Tim Kraska, and Samuel Madden. 2025. Improving DBMS Scheduling Decisions with Accurate Performance Prediction on Concurrent Queries. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4185–4198.

[67] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. 2012. PET: reducing database energy cost via query optimization. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1954–1957.

[68] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A Chien, and Haryadi S Gunawi. 2017. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *15th USENIX Conference on File and Storage Technologies*. USENIX Association, Santa Clara, CA, USA, 15–28.

[69] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a query optimizer without expert demonstrations. In *Proceedings of the 2022 International Conference on Management of Data*. 931–944.

[70] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proceedings of the VLDB Endowment* 14, 1 (2020), 61–73.

[71] Chi Zhang, Ryan Marcus, Anat Kleiman, and Olga Papaemmanouil. 2020. Buffer pool aware query scheduling via deep reinforcement learning. *arXiv preprint arXiv:2007.10568* (2020).

[72] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1658–1670.

[73] Kai Zhong, Luming Sun, Tao Ji, Cuiping Li, and Hong Chen. 2024. FOSS: A Self-Learned Doctor for Query Optimizer. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4329–4342.

[74] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query performance prediction for concurrent queries using graph embedding. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1416–1428.

[75] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1466–1479.

[76] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.

Received 17 January 2026; revised 19 May 2026; accepted 12 June 2026