NeurDB: On the Design and Implementation of an AI-powered Autonomous Database

Zhanhao Zhao[1], Shaofeng Cai[1], Haotian Gao[1], Hexiang Pan[1], Siqi Xiang[1], Naili Xing[1],
Gang Chen[2], Beng Chin Ooi[1], Yanyan Shen[3], Yuncheng Wu[4], Meihui Zhang[5]

[1]National University of Singapore [2]Zhejiang University [3]Shanghai Jiao Tong University
[4]Renmin University of China [5]Beijing Institute of Technology
{zhzhao,shaofeng,gaohaotian,xiangsq,xingnl,ooibc}@comp.nus.edu.sg,panh@u.nus.edu
cg@zju.edu.cn,shenyy@sjtu.edu.cn,wuyuncheng@ruc.edu.cn,meihui_zhang@bit.edu.cn

ABSTRACT

Databases are increasingly embracing AI to provide intelligent in-database analytics and autonomous system optimization, aiming to relieve end-user burdens across various industry sectors. Nonetheless, most existing approaches fail to account for the dynamic nature of databases, which renders them ineffective for real-world applications characterized by evolving data and workloads. This paper introduces NeurDB, an AI-powered autonomous database that deepens the fusion of AI and databases with adaptability to data and workload drift. NeurDB establishes a new in-database AI ecosystem that seamlessly integrates AI workflows within the database. This integration enables efficient and effective in-database AI analytics and fast-adaptive learned database components. Empirical evaluations demonstrate that NeurDB substantially outperforms existing solutions in managing AI analytics tasks, with the proposed learned components more effectively handling data and workload drift than state-of-the-art approaches.

1 INTRODUCTION

Database management systems (DBMSs) are becoming more intelligent and autonomous by embracing artificial intelligence (AI), catering to modern data- and AI-centric applications. AI augments DBMS functionality to support in-database AI analytics [14, 30, 45, 49], enabling complex and advanced analytics tasks such as disease progression predictions and user purchase recommendations. In parallel, AI empowers DBMSs to achieve autonomous DBMS optimization, driving a broad range of learned database designs that enhance system performance with minimal human intervention, including learned knob tuners [34], learned system drivers [21, 55], and learned database components [15] such as query optimizers [24, 54], indexes [16], and concurrency control [44], etc.

The aspiration of integrating DBMSs with AI was first expressed forty years ago [2], which has been periodically revisited with evolving technology [5, 37, 46]. With the advancements in both AI and DBMSs, considerable progress has been made in deepening their fusion. Unfortunately, a huge gap remains between the potential of this integration and its current state of usability, largely due to the inherent differences in their paradigms [28]. The dynamic nature of databases, characterized by *data and workload drift*, poses a fundamental challenge [17, 18, 48]. For example, suppose an e-commerce database stores user and product information. While the database continuously evolves due to user and merchant activities in normal periods, it would experience a sharp increase in workloads during flash sales, where huge volumes of updates from sales transactions are processed. In contrast, AI models typically derive intelligence from static datasets and thus can become outdated quickly in the face of database dynamism. For instance, models for user purchase recommendations typically acquire knowledge by training on specific user behaviors and product information. However, when the data drifts due to transactional updates, models relying on past knowledge may produce inaccurate recommendations since they are not updated to reflect new conditions. Similarly, learned query optimizers trained based on historical system environments including data distributions and workload patterns can struggle to deliver effective query plans as continuous data and workload drift.

Adapting to data and workload drift has gained traction for both in-database AI analytics and autonomous DBMS optimization. Existing databases supporting in-database AI analytics [26, 36, 40] often recommend users to completely retrain models with new data and workloads when the models become outdated. This retraining process is typically performed manually outside the database, and therefore, it may complicate the AI analytics workflow and becomes inefficient when dealing with continuous data and workload drift. To tackle this problem, in the context of autonomous DBMS optimization, recent works [18, 19, 23, 48] have started addressing the adaptability issues by automatically detecting drift and triggering model retraining. However, these approaches are often tailored to certain components. For example, approaches [18, 48] that allow learned query optimizers to withstand workload pattern changes can be ineffective for learned indexes requiring rapid adaptation to data distribution drift. Consequently, a paradigm that robustly handles data and workload drift for various learned database components remains absent.

We envision a deeper integration of AI and DBMSs for enhanced adaptability to both data and workload drift. Unlike existing approaches that merely overlay AI onto DBMSs or selectively enhance certain system components, we aim for a comprehensive fusion of AI with DBMSs that enables continuous adaptation to drift. It promises a cohesive AI-powered DBMS that provides advanced in-database AI analytics and deeply integrates AI into all its key components. However, achieving this level of integration presents significant challenges. First, the integration necessitates a

fundamental redesign of the entire AI workflow, including training, inference, and fine-tuning, within the database architecture. Second, models for AI analytics and learned database components must swiftly adapt to data and workload drift without losing effectiveness and efficiency. Therefore, developing such an AI-powered database requires both a system foundation that seamlessly supports model adaptation and AI models that are inherently adaptive in structure.

In this paper, we present NeurDB, an AI-powered autonomous DBMS that provides efficient and effective in-database AI analytics to seamlessly support modern AI-powered applications, and fully embraces AI techniques in each major system component to offer autonomous system optimization. At the core of NeurDB is an in-database AI ecosystem that deeply integrate the AI workflow into the database. In this ecosystem, we develop multiple in-database AI operators, such as model training, inference, and fine-tuning, along with an in-database AI engine to handle the execution of these operators. NeurDB can then directly support in-database AI analytics by calling these AI operators. To further simplify end-users in submitting their AI analytics tasks to NeurDB, we provide a user-friendly interface by extending SQL with PREDICT syntax. Moreover, we propose two techniques to optimize the AI ecosystem performance. First, we devise a data streaming protocol that significantly reduces data transfer overhead, yielding better performance for AI operations. Second, we develop an incremental update technique to minimize the fine-tuning cost, facilitating the fast adaptation of AI models. Based on the in-database AI ecosystem, we enable two efficient learned database components, namely a learned concurrency control algorithm and a learned query optimizer, that can adapt quickly to data and workload drift.

The remainder of the paper is structured as follows. The next section provides the necessary preliminaries and further clarifies the motivation. Section 3 describes the system overview of NeurDB, and Section 4 details the key system design, including the in-database AI ecosystem and fast-adaptive learned database components. Section 5 presents the experimental results. Section 6 discusses the related works before Section 7 concludes.

2 PRELIMINARIES AND MOTIVATION

In this section, we introduce the preliminaries of AI and DBMS integration, outline the design goals of NeurDB, and present the enhanced SQL syntax designed for in-database AI analytics.

2.1 AI and DBMS Integration

The integration of AI and DBMSs can offer each other mutual benefits [46], and building on this intuition, various research works on AI and DBMS integration have been proposed, which can be broadly categorized into two areas: in-database analytics and autonomous system optimization. Modern databases, such as Oracle [31], Microsoft Azure SQL [40], Amazon Redshift [38], and Google BigQuery [1], support in-database AI analytics, and allow AI tasks to be performed directly through user-friendly SQL interfaces. For better performance, these databases enable AI-driven knob tuning to autonomously refine database configurations.

Adapting to data and workload drift has been recognized as a critical challenge in advancing the integration of AI and DBMSs [17,

18, 48]. Many existing approaches for whether in-database analytics [14, 39] or autonomous system optimization [16, 25] depend on static models or complete retraining, limiting their ability to handle continuous data and workload drift. More importantly, these approaches are typically layered on top of DBMSs rather than being holistically embedded within DBMSs. This separation hinders real-time adaptability, because internal and fine-grained performance metrics, which are essential for detecting and addressing drift, can be difficult to capture externally [19, 44].

To mitigate these limitations, we design and develop NeurDB, an AI-powered database with AI holistically embedded from the ground up. By deeply integrating AI into all its key components and functionalities, NeurDB aims to automatically manage adaptation workflows for both in-database AI analytics and autonomous system optimization.

2.2 Design Goals

DBMSs are dynamic as data and workloads evolve over time, and therefore, the system must be designed for *adaptability* while also guaranteeing *reliability* and *scalability*. Building upon this understanding, we equip NeurDB with these three key properties.

Adaptability is the capability of a DBMS to evolve autonomously in response to drifting data and workloads. With optimal adaptability, DBMS can respond to drift in real time.

Reliability depicts the ability of a DBMS to consistently meet performance and accuracy standards, even during phases of adaptation and evolution. With optimal reliability, the system can operate at peak performance and maintain high accuracy consistently.

Scalability refers to the system's capacity to maintain or enhance performance as the workload increases by introducing more resources, such as threads or nodes.

Informally, our design goal is to ensure NeurDB can uphold reliability as quickly as possible when the system adapts or evolves due to data and workload drift, while ensuring good scalability. In addition, we design NeurDB as a general-purpose AI-powered DBMS which is expected to serve as a foundational infrastructure for advanced data- and AI-centric applications to enrich various domains with AI capabilities [28].

2.3 SQL Syntax for AI Analytics

NeurDB incorporates enhanced SQL to support AI analytics. As illustrated in Listing 1 and Listing 2, it extends from the standard SQL by introducing a PREDICT keyword to handle two typical AI tasks: regression with the VALUE OF clause and classification with the CLASS OF clause. Inspired by the original principle of SQL that allows an application developer to write a query with SELECT, and then let the DBMS find the most efficient way to store and retrieve data, we ensure that a developer can submit an AI analytics task simply with PREDICT. All the following operations, such as retrieving training data and invoking AI models, are handled automatically by NeurDB. Consequently, unlike existing solutions [1, 36] that require application developers to intervene in the execution of AI analytics, such as specifying model parameters, our approach can make the execution transparent to application developers, relieving them from the complexities of performing such analytics. Further, we plan to expose additional SQL for AI model management and other AI analytics functionalities, such as generative AI.

We now present two real-world analytics scenarios that can be directly supported by NeurDB using `PREDICT` queries.

Regression. Listing 1 shows the SQL to predict and fill in the missing scores of products based on user reviews. By specifying the features for training with `TRAIN ON` and using data stored in the 'review' table, we predict the value of the 'score' variable. Notably, for `TRAIN ON`, the asterisk automatically excludes features with unique constraints to avoid including meaningless data.

Listing 1: Syntax for a regression task

```
1  PREDICT VALUE OF score
2  FROM review
3  WHERE brand_name = 'Special_Goods'
4  TRAIN ON *
5      WITH brand_name <> 'Special_Goods'
```

Classification. The disease progression prediction in healthcare can be handed by the SQL shown in Listing 2. Similarly, this query specifies the features for training with `TRAIN ON` and the data table using `FROM` to predict the class of the 'outcome' variable, which indicates whether a patient has diabetes or not. In addition, we support directly inputting missing data using `VALUES`.

Listing 2: Syntax for a classification task

```
1  PREDICT CLASS OF outcome
2  FROM diabetes
3  TRAIN ON pregnancies, glucose, blood_pressure, ...
4  VALUES (6, 148, 72, ...), (1, 85, 66, ...), ...
```

3 SYSTEM OVERVIEW

We now describe the system architecture of NeurDB as shown in Figure 1, where we achieve the deep fusion of AI and databases by establishing an in-database AI ecosystem. Based on this AI ecosystem, we enable efficient and effective in-database AI analytics and develop fast-adaptive learned database components.

In-database AI Ecosystem. We establish the in-database AI ecosystem by holistically redesigning existing database components, such as the query executor, and introducing additional modules, including an AI engine and AI model storage. We first extend the query executor to support AI workflow. Beyond traditional operators for fetching and processing data, e.g., scan and join, the query executor in NeurDB includes in-database AI operators for model training, inference, and fine-tuning. With these AI operators, we effectively integrate the AI workflow into the database query processing. We are developing additional AI operators, such as model selection [49] and model slicing [51], to provide more comprehensive AI services. For example, a query may call the model selection operator (denoted as MSelection) to automatically select the best-suited model for a given prediction task, thereby enhancing accuracy and efficiency. We then introduce a new module, called the *AI engine*, into NeurDB. It handles the AI-related processing requests from AI operators and learned components and creates AI tasks on their behalf, where the task manager dispatches them to the CPU/GPU runtime for execution. Specifically, we propose a data streaming protocol to reduce data transfer overhead, and an incremental model update technique to facilitate fast model adaptation, thus enhancing the performance of the AI engine. We shall
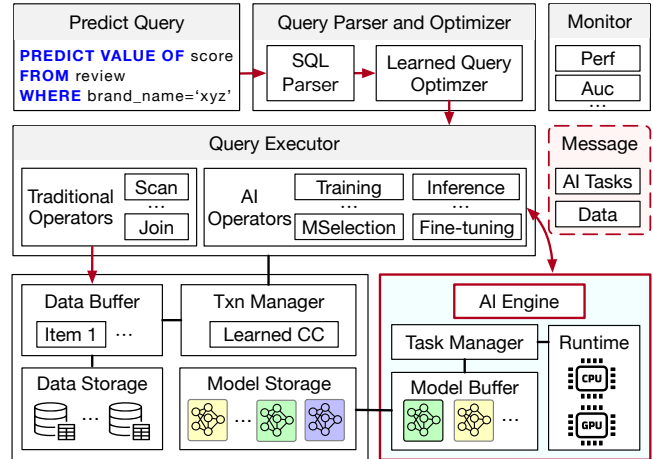

Figure 1: The System Architecture of NeurDB

elaborate on these optimizations in Section 4.1. Also, we design dedicated *AI model storage* to store AI models and serve them based on requests from the AI engine. We further implement a monitor to detect unexpected performance or accuracy issues, based on which we trigger automatic and appropriate model adaptation.

In-database AI Analytics. In Figure 1, we illustrate a running example of a `PREDICT` query. We enhance the SQL parser and the query optimizer to produce a customized query plan for `PREDICT` queries. After parsing and optimizing the query, the query executor then executes the query according to the query plan. In particular, it performs the scan operator to retrieve data and then invokes the inference operator to deliver the inference task to the AI engine. Next, the AI engine uploads the model from the model storage if it is not in the model buffer, and conducts model inference to produce the results. If the model is detected to be inaccurate, NeurDB invokes the fine-tuning operator to update the outdated model with the help of the AI engine.

Autonomous DBMS Optimization. We also enable autonomous system optimization based on the proposed in-database AI ecosystem. For instance, the learned query optimizer interacts with the AI engine to train a model responsible for generating efficient query plans. The monitor tracks the performance of the generated query plans, and if a plan is identified as inefficient, the monitor notifies the AI engine to fine-tune the model, enabling continuous optimization and adaptability. Similarly, the monitor can trigger autonomous knob tuning when suboptimal knob settings are detected, ensuring that the system remains well-configured to handle data and workload drift effectively. We will further introduce the detailed design of learned database components in Section 4.2.

4 THE DESIGN OF NeurDB

In this section, we present two key subsystems of NeurDB, namely an in-database AI ecosystem and learned database components.

4.1 In-database AI Ecosystem

AI Engine. The AI engine of NeurDB, pivotal to all AI-related activities for tasks from both user and internal learned components,
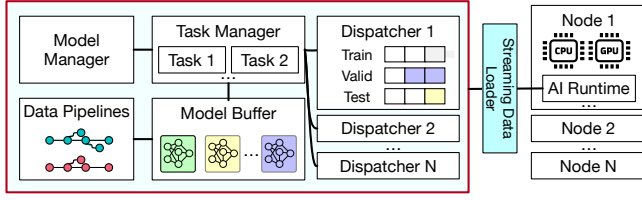
Figure 2: AI Engine of NeurDB

Figure 3: Incremental Update for Model Manager

operates on a distributed and event-driven architecture to optimize efficiency and throughput. Figure 2 shows the main components and the communicative flow with connected external nodes serving distributed AI tasks. In the AI engine, the *task manager* is the main component that coordinates and schedules the tasks and resources. It handles and parses the incoming AI tasks, and creates a *dispatcher* for each task. A dispatcher connects to multiple AI runtimes at external nodes. It also loads and caches the necessary data required by the corresponding AI tasks, performs data pipelines on it for preprocessing, feature engineering, etc, and pushes the prepared data and model weights to the remote AI runtime to trigger AI activities. Notably, the data is transferred in a streaming and pipelining manner to minimize the delay in the data preparation steps.

Data Streaming Protocol. During an AI task, the AI runtime receives data continuously from the database. NeurDB's AI engine optimizes this process with a dedicated data streaming protocol to reduce the time and memory overheads. Specifically, the AI runtime establishes a TCP socket connection with the dispatcher. When a task is assigned to the dispatcher, it first schedules the AI runtimes and performs handshakes with them to negotiate (1) model parameters, such as model structure, model arguments, the training batch size, etc, and (2) streaming parameters, e.g., the initial size for send and receive buffers and the number of batches per transmission. Then, it starts the data and model transfer through the connection. Notably, to support adaptable control and scheduling over resources, these parameters can be dynamically updated for an ongoing AI task through a data-driven dispatcher. This makes the AI engine a self-driving component, thereby controlling and optimizing its operations autonomously.

Model Manager. Given the dynamic nature of DBMSs, a typical AI lifecycle extends beyond a single model. As new data is introduced, incremental updates and retraining are required to address data and workload drift that degrade predictive performance. This process results in multiple evolving model versions, creating significant management difficulties and storage overheads. To address this challenge, NeurDB leverages the capabilities of databases to efficiently manage AI models created by either users or internal components and handle drift by design. Specifically, it introduces a dedicated *model manager* in its AI engine, enabling fine-grained model management with efficient updates. The model manager provides high-level interfaces for handling AI operations, such as training, inference, and fine-tuning, executed via *model views*. Similar to data views in DBMSs, model views serve as logical abstractions of AI models tailored for specific tasks, with physical representations maintained in model storage. Formally, a model M comprises a series of layers $L^{(j)}(j \in \mathbb{Z}^+)$. To generate outputs for data X, the layers are executed sequentially, i.e.,
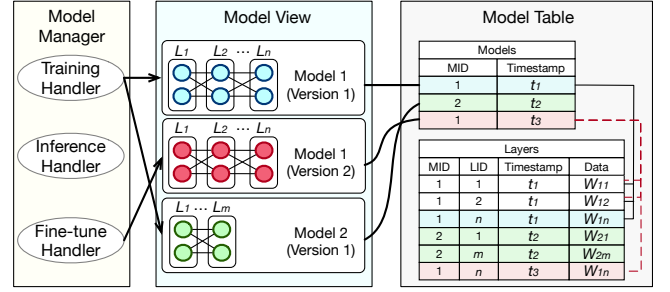
$M(X) = L^{(n)}(L^{(n-1)}(...(L^{(1)}(X))))$. This layered model storage approach aligns with the structure of deep neural networks (DNNs), ensuring fast and efficient AI model accessibility. Notably, this model formulation can also be generalized to non-linear DNN architectures that are represented as directed acyclic graphs, which can be achieved by executing layers based on the topological order.

Model Incremental Update. Leveraging its layered model storage, NeurDB enables versioning for a specific model and supports incremental model updates through fine-tuning. In particular, to adapt the current model to drifting data distributions, the AI engine selectively fine-tunes the final layers using the updated data stored in the designated database relation and freezes the preceding layers. Subsequently, only the updated layers remain persistent in the model storage, from where they can be extracted and merged with the previously frozen layers to create a new model version.

Figure 3 illustrates how the model manager supports incremental updates for AI models. Whenever an inference or fine-tuning task is initiated, the system retrieves all layers up to the most recent version to construct the model, i.e., for a model $M_{i,t}$, uniquely defined by its Model ID (MID) i and the timestamp of creation t, consisting of k layers,

$$M_{i,t}(X) = L_{i,t_k}^{(k)}(L_{i,t_{k-1}}^{(k-1)}(...(L_{i,t_1}^{(1)}(X))))$$

$$s.t. \ \forall p, q \in \{1, ..., k\}, L_{i,t_p}^{(p)} \to t_p \geq t_q \wedge t_p \leq t.$$

Let us consider the example shown in Figure 3. Supposing the model M_1 requires fine-tuning the last layer L_n, $M_{1,2}$, namely the second version of M_1, can then be assembled by layers $\{L_{1,1}^{(1)}, ..., L_{1,1}^{(n-1)}, L_{1,2}^{(n)}\}$. This allows $M_{1,1}$ and $M_{1,2}$ to share the majority of model weights, ensuring adaptability to data drift while maintaining efficient storage.

4.2 Fast-adaptive Learned Components

In this section, we introduce two key learned database components, learned concurrency control and learned query optimizer, which can achieve fast adaptation to data and workload drift. To achieve this, we fundamentally rely on the underlying AI ecosystem. We non-intrusively monitor the system conditions such as transaction throughput and data distributions, which can be used to detect data and workload drift in real time. Based on these metrics, NeurDB automatically triggers model fine-tuning to adapt to continuously evolving data and workloads, and continually generates valid input for model pre-training, allowing the model for learned database components to gain global knowledge of most drift.
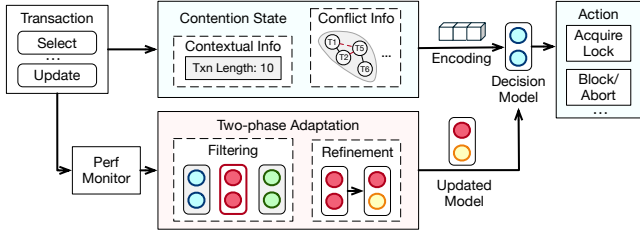
Figure 4: Fast-adaptive Learned Concurrency Control

Figure 5: Fast-adaptive Learned Query Optimizer

Learned Concurrency Control. We design an efficient learned concurrency control algorithm that can continually adapt to changing workloads, as shown in Figure 4. Given a transaction T with multiple operations, our algorithm incorporates a decision-making model $\mathcal{F}$ to assign each operation op the optimal concurrency control action δ based on the current system condition represented as x, i.e., $\delta = \mathcal{F}(x)$. Unlike state-of-the-art approach [44] that simply adjusts actions based on predefined transaction or operation patterns (e.g., transaction type), our approach learns the optimal action based on the contention state, which includes both conflict information (such as dependency) of transactions and contextual information (such as the transaction length). For instance, when a write is performed on high-contention data records, we may immediately abort the transaction to avoid unnecessary costs, as the transaction is likely to be aborted eventually. In contrast, we may execute a read on low-contention records with optimistic concurrency control without locking to avoid extra conflict detection overhead, and a long transaction that is supposed to run for a long time should have high priority. By using a modeling paradigm based on the contention state, our model is more likely to generalize to drifting workloads with varying levels of contention. However, since transactions can be completed in milliseconds, the model must be efficient so as not to become a bottleneck. To achieve this, we first develop a fast encoding technique to significantly reduce the dimension of contention state representation, and then compress the model with a flattened layer to improve inference efficiency.

As the model is compressed to trade generalizability for performance, the representation of the contention state may gradually become inaccurate over time due to workload drift. Consequently, when we detect workload drift through their impact on transaction performance, we introduce a fine-tuning process for $\mathcal{F}$ to adapt it into $\mathcal{F}_{next}$. With the leaner architecture of the model, the adaptation can be accelerated due to the narrower search space compared to a large model. Specifically, we propose a two-phase adaptation algorithm based on the online Reinforcement Learning (RL) framework. In the first filtering phase, we generate several improved models using Bayesian optimization and evaluate them over a specific timeframe to identify the best-performing model. Then, in the refinement phase, we employ reward-based feedback to further optimize the selected model.

Learned Query Optimizer. We propose a learned query optimizer that can efficiently adapt to data and workload drift, as shown in Figure 5. In contrast to existing works [24, 54] that attempt to produce the best plan for a given query Q under fixed system conditions (i.e., data distribution and workload), our approach effectively identifies the plan best suited for the current system conditions.
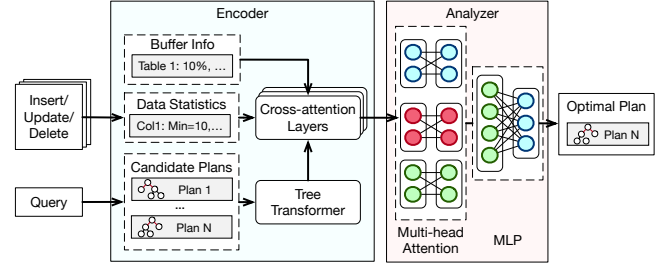
To achieve this, we design a dual-module model consisting of an encoder and an analyzer. Specifically, in the encoder, we input the vector generated by a tree transformer, which includes multiple candidate query plans for Q, along with system condition representations (including buffer information depicting buffer usage and data statistics representing each attribute's distribution), into cross-attention layers to generate a unified embedding. The analyzer then uses this embedding as input for a multi-head attention layer followed by a multilayer perceptron (MLP) to deliver the optimal plan. With more knowledge of the mapping between plans and system conditions, the model can better generalize to data and workload drift. To maximize this knowledge, we generate various synthetic data distributions and workloads using Bayesian optimization, and pre-train the model to handle most drift effectively. Consequently, the proposed learned query optimizer provides consistent query performance under evolving data and workloads.

Discussion. The main design principle we follow to develop computational models and algorithms in NeurDB is the filter-and-refine principle (FRP) [11, 28, 29]. FRP employs a two-stage strategy: a filtering stage quickly eliminates less promising or irrelevant objects (e.g., data, strategies, etc.) from a large set, followed by a more resource-intensive refinement stage on the filtered subset. This strategy enables the efficient execution of computationally demanding tasks while yielding near-optimal results. We therefore leverage FRP in both in-database AI analytics and autonomous DBMS optimization for enhanced adaptability. For example, we integrate FRP in the RL framework of learned concurrency control and incorporate it in the multi-head attention layers of the learned query optimizer. However, existing model structures, typically characterized by fixed topologies and layer configurations, constrain the full potential of FRP. We are exploring a flexible model structure that can be dynamically adjusted with the guidance of FRP, enabling seamless adaptation to data and workload drift.

5 EVALUATION

In this section, we present our preliminary evaluation results. We first introduce the experimental setup, and then evaluate the in-database AI ecosystem under real-world AI analytics scenarios, and the learned database components with data and workload drifts.

5.1 Experimental Setup

We conduct experiments on a server equipped with an Intel(R) Xeon(R) W-2133 CPU@3.60GHz (12 cores, 24 threads), 64 GB memory, and 3 NVIDIA GeForce RTX 2080 Ti GPUs. All experiments

Table 1: Queries for AI Analytics Evaluations

Workload	Statement
E-Commerce (E)	`PREDICT VALUE OF click_rate FROM avazu TRAIN ON *`
Healthcare (H)	`PREDICT CLASS OF outcome FROM diabetes TRAIN ON *`

are executed within Docker containers based on the official Ubuntu 22.04 image with CUDA 11.8.0, leveraging the host's GPU resources.

5.1.1 Benchmarks. We construct two real-world applications that require AI analytics. The AI analytics queries used in our experiments are listed in Table 1.

- **E-commerce (E) Workload** performs click-through rate prediction, a critical task in e-commerce for product recommendations, using the Avazu dataset (Avazu) [7], which consists of ~40.4M records and 22 attributes. We use k-means clustering to create five data clusters namely, C_1 to C_5, and by switching from one to another, we simulate the data distribution drift.
- **Healthcare (H) Workload** conducts disease progression prediction using the UCI Diabetes dataset (Diabetes) [9]. After scaling, the dataset comprises ~5.2M data records and 43 attributes.

We establish a **micro-benchmark** to evaluate the learned database components. It consists of a transactional benchmark based on YCSB [6], which generates synthetic workloads for large-scale Internet applications. Each transaction performs 5 selects and 5 updates on a table with 1 million records. In addition, we construct an OLAP benchmark based on the STATS dataset [8], which consists of 8 tables from the Stats Stack Exchange network. We execute inserts/updates/deletes with randomly generated data values to simulate data distribution drift following a recent work [23].

5.1.2 Implementation and Default Configuration. We have released the first version of NeurDB [27], in which we integrate our proposed in-database AI ecosystem and learned components into the codebase of PostgreSQL v16.3. We will gradually introduce new modules such as AI-powered resource scheduling, etc., and replace existing components where necessary. We implement a baseline system called PostgreSQL+P, which loads data from PostgreSQL in batches, and utilizes an AI runtime built with PyTorch to support AI analytics. By default, we employ ARM-Net [3], an adaptive relation modeling network tailored for structured data, as the basic analytics model for both PostgreSQL+P and NeurDB. In our experiments, we inherit the default settings of PostgreSQL unless otherwise specified. We set the default window size of the streaming data loader to 80 data batches. Each batch contains 4096 data records (samples).

5.2 In-database AI Analytics

We evaluate the in-database AI analytics in terms of end-to-end latency, training throughput, and loss variation with data drift.

Efficiency and Scalability. We study the efficiency of NeurDB on AI analytics workloads by comparing it with PostgreSQL+P. As observed in Figure 6(a), NeurDB achieves up to 41.3% and 48.6% lower end-to-end latency, and 1.96× and 2.92× higher training throughput than PostgreSQL+P for Workload E and Workload H, respectively. The significant improvement in latency and throughput achieved by NeurDB is due to its in-database AI ecosystem, which efficiently supports AI analytics by utilizing a data streaming protocol. Further,
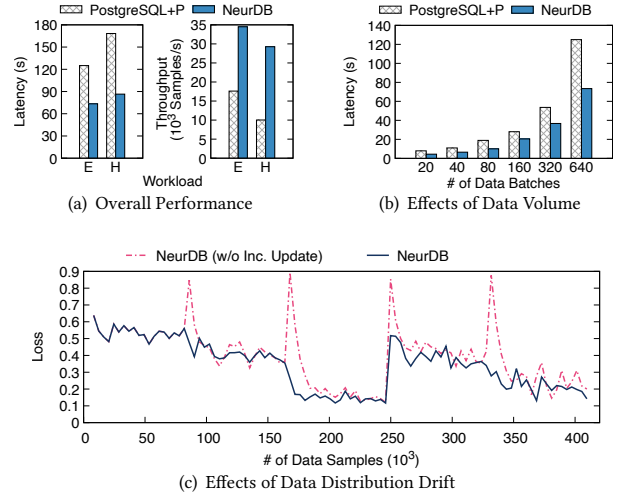

(a) Overall Performance (b) Effects of Data Volume

(c) Effects of Data Distribution Drift

Figure 6: End-to-end Performance of AI Analytics

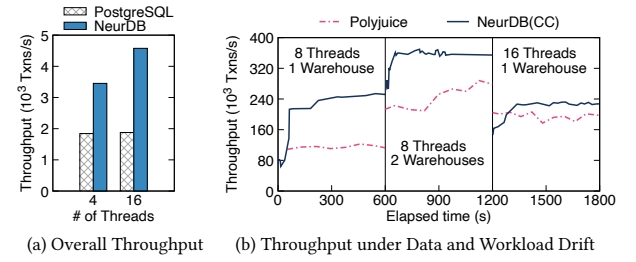(a) Overall Throughput (b) Throughput under Data and Workload Drift

Figure 7: Performance of Learned Concurrency Control

to evaluate the impact of data volume on the end-to-end latency, we run Workload E with varying numbers of data batches. As shown in Figure 6(b), NeurDB consistently outperforms PostgreSQL+P, indicating that NeurDB can scale well with increased data volume.

Adaptability. We investigate NeurDB's ability to adapt to the drifting data and workloads with the model incremental update technique. To simulate such drift, for $i \in [1, 4]$, we let NeurDB perform the training task using cluster C_i of Workload E, and switch to C_{i+1} when 81,920 samples of C_i are consumed by model training. Figure 6(c) plots the training losses with and without model incremental updates. From the result, we can observe that starting from the first data drift, the AI engine equipped with incremental updates receives lower loss values during the sudden drift in data distributions. This enables the model to converge faster, and as a result, NeurDB is equipped to serve the new tasks effectively.

5.3 Learned Database Components

We now investigate the performance of our proposed learned database components.

Learned Concurrency Control. To evaluate the proposed concurrency control algorithm, we compare NeurDB with PostgreSQL using the micro-benchmark with varying thread counts. The results, shown in Figure 7(a), demonstrate that NeurDB achieves up
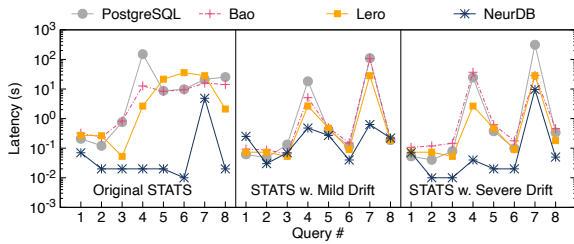
Figure 8: Performance of Learned Query Optimizers

to 1.44× higher transaction throughput than PostgreSQL. We attribute this performance gain to the proposed algorithm's ability to schedule transactions more effectively than serializable snapshot isolation [35], a static concurrency control algorithm employed in PostgreSQL. We further evaluate the adaptability of our proposed algorithm against Polyjuice [44], state-of-the-art learned algorithm. Due to the limitation of constraining the transaction execution workflow, implementing Polyjuice on the codebase of NeurDB would be cumbersome. We therefore opt to implement our algorithm, named NeurDB(CC), into the Polyjuice codebase to facilitate a fair comparison. We set up a drift workload based on TPCC [42] by varying the number of warehouses and threads. As shown in Figure 7(b), NeurDB(CC) adapts quickly to workload drift and outperforms Polyjuice by up to 2.05×. The superior performance of NeurDB(CC) mainly stems from its design that encapsulates a fast yet accurate model to find the best concurrency control action, while facilitating the fine-tuning process with the two-phase adaptation algorithm.

Learned Query Optimizer. We next compare NeurDB with PostgreSQL and two state-of-the-art learned query optimizer approaches, namely Bao [24], and Lero [54]. We construct three workloads with different data distributions and randomly select 8 SPJ queries provided by STATS datasets. We use stable models of Bao and Lero for the experiment, as they demonstrated good performance in their respective papers.

As can be observed in Figure 8, NeurDB achieves up to 20.32% lower average latency of all evaluated queries, which demonstrates its effective adaptability to both data and workload drift. Due to the proposed dual-module model, NeurDB is able to effectively capture system conditions and use them to select an efficient query plan.

6 RELATED WORK

Our work relates to a broad spectrum of efforts on the fusion of AI and databases. The initial concepts [2, 12] can be traced back to the 1980s. At that time, the fusion was far from reality due to limited development in both realms. With advancements in AI and database fields over the years, numerous attempts by academia [15, 34] and industry [14, 21] have been made to advance in-database AI analytics and autonomous DBMS optimization. However, many problems remain open, and it is important to consider data and workload drift in both directions. Here we shall briefly discuss recent advancements in AI and DBMS integration, with a particular focus on their approaches to handling data and workload drift.

In-database AI Analytics. Recent research increasingly focuses on optimizing the performance of in-database AI analytics. For example, some studies [13, 14, 33, 39] extend the existing query and transaction execution framework to improve the efficiency of analytics tasks. Further, in-database model management systems [41, 43, 53] are proposed to minimize model storage costs while efficiently serving models for analytics tasks. These optimizations can be broadly adopted by full-fledged DBMSs supporting AI analytics, such as nsDB [50], GaussML [20], and NeurDB. However, apart from the mentioned approaches, NeurDB establishes an in-database AI ecosystem, specifically designed to enhance the efficiency and effectiveness of AI analytics under continuous data and workload drift.

Autonomous DBMS Optimization. Traditional AI-driven DBMS optimization in areas such as knob tuning [34], resource management [21, 22], and learned system components [15, 16, 24, 44, 54] typically relies on complete retraining to handle data and workload drift. Recently, the development of drift-handling mechanisms become popular, particularly for learned query optimizers and learned indexes. For example, some works introduce methods to automatically detect drift and trigger model updates [18, 19, 48], enabling learned query optimizers to withstand data and workload drift. Further, updatable learned indexes [10, 47, 52] are enhanced with the adaptability to continuous transactional updates. Unlike existing works focusing on certain components, NeurDB introduces a unified architecture based on the proposed in-database AI ecosystem to facilitate fast adaptation across all its key components.

We are extending NeurDB to be an intelligent cloud-native HTAP database with built-in support for advanced AI analytics. In particular, we disaggregate the system architecture into separate compute and storage layers [4, 32], enabling dedicated compute nodes to handle OLTP, OLAP, and AI analytics independently. Based on this foundation, we are devising AI-powered resource scheduling to enable on-demand resource allocation and component assembling, thus enhancing system scalability and adaptability.

7 CONCLUSIONS

This paper presents NeurDB, a novel AI-powered autonomous DBMS that is adaptable to data and workload drift. NeurDB, with its fast-adaptive learned database components and in-database AI ecosystem, facilitates efficient and effective in-database AI analytics and autonomous system optimization. Empirical evaluations demonstrate the superiority of NeurDB, highlighting its potential to realize a seamless fusion of AI and databases.

REFERENCES

[1] Google BigQuery. 2024. https://cloud.google.com/bigquery/docs.
[2] Michael L. Brodie. 1988. Future Intelligent Information Systems: AI and Database Technologies Working Together. In *AAAI*. AAAI Press / The MIT Press, 844–845.
[3] Shaofeng Cai, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. 2021. ARM-Net: Adaptive Relation Modeling Network for Structured Data. In *SIGMOD Conference*. ACM, 207–220.
[4] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *SIGMOD Conference*. ACM, 2477–2489.
[5] Gao Cong, Jingyi Yang, and Yue Zhao. 2024. Machine Learning for Databases: Foundations, Paradigms, and Open problems. In *SIGMOD Conference Companion*. ACM, 622–629.
[6] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *SoCC*. ACM, 143–154.
[7] Avazu Dataset. 2024. https://www.kaggle.com/c/avazu-ctr-prediction.
[8] STATS Dataset. 2024. https://github.com/Nathaniel-Han/End-to-End-CardEst-Benchmark.
[9] UCI Diabetes Dataset. 2024. https://archive.ics.uci.edu/ml/datasets.
[10] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *SIGMOD Conference*. ACM, 969–984.
[11] Filter and Refine. 2024. https://en.wikipedia.org/wiki/Filter_and_refine/.
[12] Goetz Graefe and David J. DeWitt. 1987. The EXODUS Optimizer Generator. In *SIGMOD Conference*. ACM Press, 160–172.
[13] Matthias Jasny, Tobias Ziegler, Tim Kraska, Uwe Röhm, and Carsten Binnig. 2020. DB4ML - An In-Memory Database Kernel with Machine Learning Support. In *SIGMOD Conference*. ACM, 159–173.
[14] Konstantinos Karanasos, Matteo Interlandi, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Doris Xin, Supun Nakandala, Subru Krishnan, Markus Weimer, Yuan Yu, Raghu Ramakrishnan, and Carlo Curino. 2020. Extending Relational Query Processing with ML Inference. In *CIDR*. www.cidrdb.org.
[15] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB: A Learned Database System. In *CIDR*. www.cidrdb.org.
[16] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD Conference*. ACM, 489–504.
[17] Meghdad Kurmanji, Eleni Triantafillou, and Peter Triantafillou. 2024. Machine Unlearning in Learned Databases: An Experimental Analysis. *Proc. ACM Manag. Data* 2, 1 (2024), 49:1–49:26.
[18] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. *Proc. ACM Manag. Data* 1, 1 (2023), 33:1–33:27.
[19] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *SIGMOD Conference*. ACM, 1920–1933.
[20] Guoliang Li, Ji Sun, Lijie Xu, Shifu Li, Jiang Wang, and Wen Nie. 2024. GaussML: An End-to-End In-Database Machine Learning System. In *ICDE*. IEEE, 5198–5210.
[21] Guoliang Li, Xuanhe Zhou, Ji Sun, Xiang Yu, Yue Han, Lianyuan Jin, Wenbo Li, Tianqing Wang, and Shifu Li. 2021. openGauss: An Autonomous Database System. *Proc. VLDB Endow.* 14, 12 (2021), 3028–3041.
[22] Ji-You Li, Jiachi Zhang, Wenchao Zhou, Yuhang Liu, Shuai Zhang, Zhuoming Xue, Ding Xu, Hua Fan, Fangyuan Zhou, and Feifei Li. 2023. Eigen: End-to-end Resource Optimization for Large-Scale Databases on the Cloud. *Proc. VLDB Endow.* 16, 12 (2023), 3795–3807.
[23] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. *Proc. VLDB Endow.* 17, 2 (2023), 197–210.
[24] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD Conference*. ACM, 1275–1288.
[25] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718.
[26] MindsDB. 2024. https://mindsdb.com/.
[27] NeurDB. 2024. https://github.com/neurdb/neurdb.
[28] Beng Chin Ooi, Shaofeng Cai, Gang Chen, Yanyan Shen, Kian-Lee Tan, Yuncheng Wu, Xiaokui Xiao, Naili Xing, Cong Yue, Lingze Zeng, Meihui Zhang, and Zhanhao Zhao. 2024. NeurDB: An AI-powered Autonomous Data System. *SCIENCE CHINA Information Sciences* 67, 200901 (2024). https://doi.org/10.1007/s11432-024-4125-9

[29] Beng Chin Ooi, HweeHwa Pang, Hao Wang, Limsoon Wong, and Cui Yu. 2002. Fast Filter-and-Refine Algorithms for Subsequence Selection. In *IDEAS*. IEEE Computer Society, 243–255.
[30] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A Distributed Deep Learning Platform. In *ACM Multimedia*. ACM, 685–688.
[31] Oracle. 2024. https://docs.oracle.com/en/database/oracle/machine-learning/.
[32] Xi Pang and Jianguo Wang. 2024. Understanding the Performance Implications of the Design Principles in Storage-Disaggregated Databases. *Proc. ACM Manag. Data* 2, 3 (2024), 180.
[33] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end Optimization of Machine Learning Prediction Queries. In *SIGMOD Conference*. ACM, 587–601.
[34] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *CIDR*. www.cidrdb.org.
[35] Dan R. K. Ports and Kevin Grittner. 2012. Serializable Snapshot Isolation in PostgreSQL. *Proc. VLDB Endow.* 5, 12 (2012), 1850–1861.
[36] PostgresML. 2024. https://postgresml.org/.
[37] Christopher Ré, Divy Agrawal, Magdalena Balazinska, Michael J. Cafarella, Michael I. Jordan, Tim Kraska, and Raghu Ramakrishnan. 2015. Machine Learning and Databases: The Sound of Things to Come or a Cacophony of Hype?. In *SIGMOD Conference*. ACM, 283–284.
[38] Amazon Redshift. 2024. https://docs.aws.amazon.com/redshift/latest/dg/machine_learning.html.
[39] Ricardo Salazar-Díaz, Boris Glavic, and Tilmann Rabl. 2024. InferDB: In-Database Machine Learning Inference Using Indexes. *Proc. VLDB Endow.* 17, 8 (2024), 1830–1842.
[40] Azure SQL. 2024. https://learn.microsoft.com/en-us/azure/azure-sql/.
[41] Zhaoyuan Su, Ammar Ahmed, Zirui Wang, Ali Anwar, and Yue Cheng. 2024. Everything You Always Wanted to Know About Storage Compressibility of Pre-Trained ML Models but Were Afraid to Ask. *Proc. VLDB Endow.* 17, 8 (2024), 2036–2049.
[42] TPCC. 2024. http://www.tpc.org/tpcc/.
[43] Manasi Vartak. 2017. MODELDB: A System for Machine Learning Model Management. In *CIDR*. www.cidrdb.org.
[44] Jia-Chen Wang, Ding Ding, Huan Wang, Conrad Christensen, Zhaoguo Wang, Haibo Chen, and Jinyang Li. 2021. Polyjuice: High-Performance Transactions via Learned Concurrency Control. In *OSDI*. USENIX Association, 198–216.
[45] Wei Wang, Gang Chen, Tien Tuan Anh Dinh, Jinyang Gao, Beng Chin Ooi, Kian-Lee Tan, and Sheng Wang. 2015. SINGA: Putting Deep Learning in the Hands of Multimedia Users. In *ACM Multimedia*. ACM, 25–34.
[46] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Rec.* 45, 2 (2016), 17–22.
[47] Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, and Chunxiao Xing. 2021. Updatable Learned Index with Precise Positions. *Proc. VLDB Endow.* 14, 8 (2021), 1276–1288.
[48] Peizhi Wu and Zachary G. Ives. 2024. Modeling Shifting Workloads for Learned Database Systems. *Proc. ACM Manag. Data* 2, 1 (2024), 38:1–38:27.
[49] Naili Xing, Shaofeng Cai, Gang Chen, Zhaojing Luo, Beng Chin Ooi, and Jian Pei. 2024. Database Native Model Selection: Harnessing Deep Neural Networks in Database Systems. *Proc. VLDB Endow.* 17, 5 (2024), 1020–1033.
[50] Ye Yuan, Bo Tang, Tianfei Zhou, Zhiwei Zhang, and Jianbin Qin. 2024. nsDB: Architecting the Next Generation Database by Integrating Neural and Symbolic Systems (Vision). *Proc. VLDB Endow.* 17, 11 (2024), 3283–3289.
[51] Lingze Zeng, Naili Xing, Shaofeng Cai, Gang Chen, Beng Chin Ooi, Jian Pei, and Yuncheng Wu. 2025. Powering In-Database Dynamic Model Slicing for Structured Data Analytics. *Proc. VLDB Endow.* 17, 1 (2025), 1020–1033.
[52] Shunkang Zhang, Ji Qi, Xin Yao, and André Brinkmann. 2024. Hyper: A High-Performance and Memory-Efficient Learned Index via Hybrid Construction. *Proc. ACM Manag. Data* 2, 3 (2024), 145.
[53] Lixi Zhou, Jiaqing Chen, Amitabh Das, Hong Min, Lei Yu, Ming Zhao, and Jia Zou. 2022. Serving Deep Learning Models with Deduplication from Relational Databases. *Proc. VLDB Endow.* 15, 10 (2022), 2230–2243.
[54] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A Learning-to-Rank Query Optimizer. *Proc. VLDB Endow.* 16, 6 (2023), 1466–1479.
[55] Rong Zhu, Lianggui Weng, Wenqing Wei, Di Wu, Jiazhen Peng, Yifan Wang, Bolin Ding, Defu Lian, Bolong Zheng, and Jingren Zhou. 2024. PilotScope: Steering Databases with Machine Learning Drivers. *Proc. VLDB Endow.* 17, 5 (2024), 980–993.