A Unified Transferable Model for ML-Enhanced DBMS

Ziniu Wu[2], Pei Yu[1,4,#], Peilun Yang[1,3,#], Rong Zhu[1,*], Yuxing Han[1], Yaliang Li[1],
Defu Lian[4], Kai Zeng[1], Jingren Zhou[1]

[1]Alibaba Group, [2]Massachusetts Institute of Technology,
[3]University of Technology Sydney, [4]University of Science and Tehnology, China

ziniuw@mit.edu, {yangpeilun.ypl, yupei.yu, red.zr, yaliang.li, zengkai.zk, jingren.zhou}@alibaba-inc.com, yxhan.me@gmail.com, liandefu@ustc.edu.cn

ABSTRACT

Recently, the database management system (DBMS) community has witnessed the power of machine learning (ML) solutions for DBMS tasks. Despite their promising performance, these existing solutions can hardly be considered satisfactory. First, these ML-based methods in DBMS are not effective enough because they are optimized on each specific task, and cannot explore or understand the intrinsic connections between tasks. Second, the training process has serious limitations that hinder their practicality, because they need to retrain the entire model from scratch for a new DB. Moreover, for each retraining, they require an excessive amount of training data, which is very expensive to acquire and unavailable for a new DB. We propose to explore the transferabilities of the ML methods both across tasks and across DBs to tackle these fundamental drawbacks.

In this paper, we propose a unified model MTMLF that uses a multi-task training procedure to capture the transferable knowledge across tasks and a *pre-train fine-tune* procedure to distill the transferable meta knowledge across DBs. We believe this paradigm is more suitable for cloud DB service, and has the potential to revolutionize the way how ML is used in DBMS. Furthermore, to demonstrate the predicting power and viability of MTMLF, we provide a concrete and very promising case study on query optimization tasks. Last but not least, we discuss several concrete research opportunities along this line of work.

1. INTRODUCTION

Database management system (DBMS) is the cornerstone of a broad range of applications such as big data platforms, cloud computing, internet of things, and artificial intelligence. Designing and tuning DBMS involves a series of complicated tasks ranging from physical design, configuration tuning, to query optimization and execution scheduling, which all require intensive expertise. With the growth of data volume and complexity, it becomes increasingly difficult to maintain DBMS purely using human efforts.

Recently, the prosperity of machine learning (ML), especially deep learning, helps to resolve a large number of DBMS challenges. ML techniques enable automatic, fine-grained, and more accurate characterization of the problem space and benefit a variety of tasks in DBMS. Specifically, unsupervised ML techniques can model the data distribution for cardinality estimation (CardEst) [Zhu et al. 2021, Hilprecht et al. 2020, Wu and Shaikhha 2020, Yang et al. 2019, Yang et al. 2020] and indexing [Kraska et al. 2018, Ding et al. 2020b, Ding et al. 2020a, Nathan et al. 2020]; supervised ML models can replace the cost estimator (CostEst) [Sun and Li 2019, Marcus and Papaemmanouil 2019, Siddiqui et al. 2020] and execution scheduler [Marcus and Papaemmanouil [n.d.], Sheng et al. 2019]; and reinforcement learning methods solve decision making problems such as configuration tuning [Zhang et al. 2019, Li et al. 2019, Basu et al. 2016] and join order selection (JoinSel) [Marcus et al. 2019, Marcus and Papaemmanouil 2018, Guo and Daudjee 2020, Yu et al. 2020, Ortiz et al. 2018].

Motivation: Despite these ML methods' promising results on each individual task, the existing ML techniques in DBMS do not explore the following *transferabilities* and can lead to impractical solutions and/or ineffective models.

(1) *Transferability across databases*: Existing ML methods for DBMS only focus on learning the *database-specific knowledge* and ignore the *database-agnostic meta knowledge* that can be transferred to new DBs. Therefore, they need to retrain the entire model from scratch for a new DB, and generally require an excessive and impractical amount of data, such as executed queries and logs, for each retraining, which is very expensive to acquire especially for a new DB [Ma et al. 2020] (referred to as the notorious "cold-start" problem). Fortunately, some *meta knowledge* can be distilled and shared across DBs to mitigate this problem. This knowledge (such as expert experience and heuristics in the physical join implementation and access path selection) is independent of each specific DB. For example, the query optimizer usually chooses an index scan for high-selectivity predicates and a sequential scan for low-selectivity ones; and the hash join is usually more memory-intensive than nest loop join and merge join. This knowledge should be distilled and shared across various databases to avoid the redundant learning process and mitigate the "cold-start" problem.

(2) *Transferability across tasks*: Existing ML approaches are only optimized on individual DBMS tasks and neglect

# These two authors contribute equally to this paper.
∗ Corresponding author.

12th Annual Conference on Innovative Data Systems Research (CIDR '22) January 9-12, 2022, Chaminade, USA.

the *task-shared knowledge*, leading to inefficient use of data and ineffective model. Since all these approaches are fundamentally based on understanding the data distributions and query workload representation, the shared knowledge can be used to reduce model redundancy and improve data efficiency across tasks. More importantly, it can enhance the model effectiveness because these tasks are inter-dependent and this knowledge can capture the inherent interactions. For example, the purpose of CardEst model is to help generate better query plans. However, different estimations have different impacts on the quality of the generated plan, which is also determined by the plan enumeration method and the cost model. Sometimes a series of bad estimations will not lead to a worse plan, but a small estimation error of a critical sub-query can have catastrophic outcomes. Thus, a CardEst model trained without considering other tasks can not effectively generate better query plans [Negi et al. 2021].

Inspired by the recent success of the pre-trained models (e.g., BERT [Devlin et al. 2018] and GPT-3 [Brown et al. 2020]) in NLP domain, we advocate for the next generation of ML-based methods for DBMS to explore and exploit the aforementioned *transferabilities* in a unified framework. Within this framework, the knowledge can be distilled and shared across tasks to mutually benefit all, and the meta knowledge can be reused for new DBs. Specifically, we propose a meta-learning paradigm that pre-trains a model on various DBs to condense the *database-agnostic meta knowledge* and fine-tunes this model to fit a new DB with a small number of training examples. Moreover, we propose a multi-task training procedure that simultaneously trains the model on all DBMS tasks to extract the *task-shared knowledge*.

Our Contributions: We identify the *transferable* and *non-transferable* knowledge that ML models try to understand and use to solve the DBMS tasks. Based on the *transferability* across DBs, we classify the knowledge into *database-agnostic meta knowledge* and *database-specific knowledge*. Based on the *transferability* across tasks, we classify the knowledge into *task-shared knowledge* and *task-specific knowledge*.

Thereafter, we propose the multi-task meta-learning framework (MTMLF) with three modules: (1) a featurization and encoding module to characterize the *database-specific knowledge* such as the data distributions in each DB, (2) a shared representation module to extract *task-shared knowledge* that would benefit all DBMS tasks, and (3) a task-specific module to tackle each task (such as CardEst, CostEst, JoinSel, indexing, and configuration tuning) and learn the *task-specific knowledge*. Furthermore, the architecture of MTMLF naturally enables a *pre-train fine-tune* meta learning paradigm to distill the *database-agnostic meta knowledge*.

In order to demonstrate the viability of the envisioned MTMLF, we provide a case study *MTMLF-QO* for query optimization tasks, including CardEst, CostEst, and JoinSel. Thanks to the multi-task joint learning, the *MTMLF-QO* on a single DB outperforms the previous state-of-the-art (SOTA) method on CardEst/CostEst tasks, and yields near-optimal results in JoinSel task. When trained on multiple DBs with the proposed meta-learning algorithms, *MTMLF-QO* has ability to distill the *meta knowledge* that can be transferred on new DBs.

The contributions of this paper are summarized as follows:

1. We identify and classify the knowledge that ML models in DBMS essentially trying to comprehend (Section 2).
2. We propose the MTMLF, a unified transferable model for all DBs and all tasks in DBMS (Section 2).
3. We design a concrete model MTMLF-QO to showcase that MTMLF's viability for query optimization (Section 3).
4. We conduct experiments to demonstrate MTMLF-QO's superior performance and effectiveness of multi-task learning and multi-DB meta-learning (Section 4).
5. We point out several concrete future research directions along this line of work (Section 5).

2. MULTI-TASKING META-LEARNING FRAMEWORK

In this section, we classify the knowledge that ML methods in DBMS trying to comprehend, from the data and task dimensions in Section 2.1. Based on this classification, we design the multi-tasking meta-learning framework (MTMLF) to explicitly learn each type of knowledge in Section 2.2. At last, we provide the workflow about the MTMLF with the potential to revolutionize the way how ML methods are used for DBMS in Section 2.3.

2.1 Knowledge classification

The ML solutions for DBMS are fundamentally based on extracting knowledge from the DB and apply it to various tasks. We can classify this knowledge from two aspects as shown in Figure 1.

From data aspect based on whether the knowledge is transferable across DBs, it can be classified into *database-specific* and *database-agnostic meta* knowledge:

Database-specific knowledge refers to the knowledge that is unique and can hardly benefit other DBs. Specifically, it includes the data distributions, the join schema (i.e. the fact/dimension tables and their join relationship), and the query workload in a DB.

Database-agnostic meta knowledge refers to the knowledge that should be distilled and shared across various DBs. In a high level, this knowledge is independent of the data distributions and query workloads in specific DBs, such as the expert experience and heuristics about the physical join implementation and access path selection. E.g., to implement a hash join for foreign key join, the dimension table is usually the build side and the fact table is the probe side. Furthermore, in a distributed setting, the hash join is usually implemented using broadcast join where the dimension table is broadcasted. This type of meta knowledge should be shared across DBs to avoid redundant learning process for each new DB.

From task aspect based on the knowledge's transferability across tasks, it can be classified as *task-shared* and *task-specific* knowledge:

Task-shared knowledge refers to the data and query representation that can benefit all tasks in DBMS. The existing ML approaches to all DBMS tasks are fundamentally based on understanding the underlying data distributions and query workload representation. Therefore, these tasks are inter-dependent and the shared knowledge can capture their inherent interactions to enhance the model effectiveness. For instance, the index recommender analyzes the

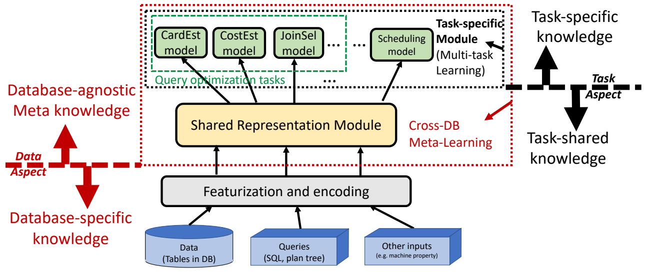

Figure 1: Framework and knowledge overview

data and query workload to recommend an index that can improve a large portion of join and scan cases encountered by the query optimizer. Conversely, the query optimizer, which plans the execution essentially based on understanding the distributions, can generate more efficient query plans by considering the learned index.

Task-specific knowledge will be used to tackle each specific task based on the shared data and query workload representation. For example, having access to the shared representation, a JoinSel model still needs to explicitly solve an NP-hard problem [Leis et al. 2015]. Specifically, after deriving estimated cardinalities and the cost of different operations from the shared knowledge, the JoinSel model will design specific features to solve an NP-hard combinatorial optimization problem and plan an optimal join order. This step is hardly beneficial to other tasks and too complex to be shared with.

Existing ML approaches in DBMS are not effective or practical mainly because they did not explore or learn the *database-agnostic meta knowledge* and *task-specific knowledge*. Therefore, we propose the MTMLF to explicitly learn these two types of knowledge.

2.2 Framework overview

We design the MTMLF with the following architecture to explicitly capture the aforementioned four types of knowledge (shown in Figure 1). First, the MTMLF uses a *featurization and encoding module* for each DB to process the non-transferable *database-specific knowledge*. Second, it constructs a *shared representation module* to learn the data distributions and query workload representation that can benefit all DBMS tasks. This module will be trained jointly on all tasks in order to extract the *task-shared knowledge* and improve model effectiveness on each task. Third, it creates a *task-specific module* with each sub-module corresponding to one DBMS task and understands the *task-specific knowledge*. Fourth, the architecture design of MTMLF enables an effective meta-learning procedure to distill the *database-agnostic meta knowledge*. All database-specific information is pushed to the *featurization and encoding module*. The remaining modules are devoted to understanding the *database-agnostic knowledge*. Therefore, we hypothesize that the *task-specific* and *shared representation modules* can benefit significantly from *pre-train fine-tune* paradigm. I.e. we can pre-train these two modules of MTMLF using data from various DBs; thereafter, when deployed on a new DB, the pre-trained model only needs a small number of training examples to fine-tune for the best accuracy.

Featurization and encoding module adaptively applies feature engineering to three types of input: the data tables, the executed query workload, and additional information such as join schema and physical machine properties. (1) This module will take each data table in the DB as input and output its encoded distribution. (2) This module can directly apply the existing procedures [Sun and Li 2019, Marcus et al. 2019] to featurize each query in the workload. (3) Some tasks may take additional information into consideration. For example, a resource allocation and execution scheduling model might need to know the memory size, buffer size, CPU usages, etc. This module can also featurize these inputs accordingly.

Shared representation module takes the featurized and encoded inputs, models their interactions, and outputs a shared representation that could benefit all tasks/DBs. For example, many tasks (e.g. CardEst, JoinSel) must understand the data distribution of the join on multiple tables. This module can learn such distribution by analyzing the cardinality of executed join queries and combining the single table distributions. Inspired by recent advance using pre-train models for NLP [Brown et al. 2020, Devlin et al. 2018], data cleaning [Tang et al. 2020] and relational table understanding [Herzig et al. 2020, Deng et al. 2020], we advocate for implementing the shared representation module with transformer [Vaswani et al. 2017], which is demonstrated to be very powerful in modeling interactions, extracting effective representations, and easy for *pre-train fine-tune* procedure.

Task-specific module contains a series of models corresponding to all DBMS tasks, some of which may contain many sub-tasks (for example, the query optimization task consists of CardEst, CostEst, JoinSel sub-tasks). Each model takes the shared representation from the previous module and returns the desired outputs for its corresponding DBMS task. This module learns the *task-specific knowledge*, which can also benefit various DBs through meta-learning.

2.3 Workflow overview

The MTMLF has the potential to revolutionize the way how ML is used in DBMS. It is more suitable in the form of cloud service, which can significantly reduce the time and complexity of adopting ML-powered DBMS components (such as DB auto-tuner and learned query optimizer) on users' DBs, and boost the wide applications of these ML components. We provide the details from the service provider and the users sides.

Service provider side: The cloud service provider will train the MTMLF on multiple users' DBs and provide the *shared representation* and *task-specific* modules as part of the DB service to the users. In this way, the provider can leverage its advantages: 1) it has access to various users' DBs either through anonymous access or federated learning [Konečný et al. 2015] to protect the data privacy; 2) it has powerful computation resources to train large models; and 3) it has abundant time because the training process is offline. Thus, the pre-trained MTMLF can fully exploit these advanatges to distill the *database-agnostic meta knowledge* beneficial to all users' DBs.

Furthermore, the service provider can periodically collect useful information from the users side in the form of anonymous training data or gradients of model parameters (in federated learning). This information will be used continuously and asynchronously to update and optimize the pre-trained

MTMLF. The new model will be published as service upgrades to benefit all users.

Users side: The DB users will locally adjust the received pre-trained MTMLF to best fit their DBs. This fast local training process only requires the users to (1) analyze the data tables in the user's DB to summarize the data distributions, similar to an "ANALYZE" operation in traditional DBMS [Group 2018], and (2) execute a small number of representative queries to fine-tune the pre-trained MTMLF. This training procedure only needs to be conducted once and all tasks are tuned to the best performance.

The MTMLF model is very effective in inference. Since all tasks are trained jointly to learn the task inherent interactions, the inference of each task can effectively take others into consideration, guaranteed to make consistent decisions. For example, the physical design wizard will only recommend indices that the query optimizer finds useful for a large portion of query workloads.

This *pre-train fine-tune* paradigm can significantly reduce the management complexity. First, the MTMLF can swiftly evolve itself as the DB changes. When the data or query workload distribution in this DB shifts, only the *featurization and encoding module* of MTMLF needs to be updated without affecting the other two modules. Second, despite the diverse set of DBMS tasks, only a single model needs to be maintained, monitored, and regularly updated.

3. CASE STUDY: QUERY OPTIMIZATION

In this section, we describe the MTMLF in a concrete scenario, query optimization (QO), a key component in DBMS. In Section 3.1, we first review the relevant learning tasks in QO: CardEst, CostEst, and JoinSel tasks. Then in Section 3.2, we present the case study model, MTMLF-QO for a single DB, which can be jointly trained on these three tasks to mutually benefit all. At last in Section 3.3, we explain the meta-learning algorithm of MTMLF-QO for multiple DBs, which helps distill the meta-knowledge beneficial to all DBs.

3.1 Learning tasks in query optimization

The query optimizer, which takes as input a SQL query and outputs its physical execution plan, directly determines the performance of DBMS. Tuning the query optimizer is a challenging task, requiring thousands of expert-engineering hours [Marcus et al. 2019, Leis et al. 2015]. Thus, numerous efforts have been devoted to optimizing QO using ML techniques [Zhou et al. 2020].

Each candidate plan of a query Q can be regarded as a tree, where each leaf node is a (sequential or index) scan operation on some tables and each inner node is a (merge, nested loop, or hash) join operation between multiple tables. Following previous work [Marcus et al. 2019], as we focus on JoinSel, we omit other physical operations (e.g. aggregate or hash). The QO process would enumerate several candidate plans, estimate their cardinality and cost, and select the optimal one. Next, we list the core learning tasks of QO:

- *Cardinality estimation (CardEst)* refers to estimating the number of tuples satisfying a query before its execution. ML-based CardEst techniques try to build either unsupervised models characterizing the data distribution [Hilprecht et al. 2020] or supervised models mapping featurized queries to the cardinality [Kipf et al. 2019]. Recent evaluation results [Han et al. 2021] have exhibited their superiority over traditional methods.
- *Cost estimation (CostEst)* refers to estimating the latency and/or throughput of a (sub-)query execution plan. ML-based CostEst methods use tree-based models (such as tree convolution [Marcus and Papaemmanouil 2019] and tree-LSTM [Sun and Li 2019]) to encode a plan and map the encoding to its estimated costs.
- *Join order selection (JoinSel)* decides the order with minimal cost to join multiple tables in the query. It is an NP-hard problem with a large search space [Leis et al. 2015]. Existing ML-based solutions attempt to effectively solve JoinSel using deep reinforcement learning techniques [Marcus et al. 2019, Marcus and Papaemmanouil 2018, Guo and Daudjee 2020, Yu et al. 2020].

These core tasks in QO are interdependent. Specifically, CostEst is fundamentally based on CardEst; JoinSel requires CardEst and CostEst to evaluate the quality of the join order. It has been shown empirically that the CardEst model learned without considering the join order and cost model will not generate effective prediction [Negi et al. 2021]. Thus, these tasks will be learned jointly in MTMLF-QO.

3.2 Architecture

As a concrete case study of the aforementioned MTMLF, the MTMLF-QO model also consists of inputs (I), featurization and encoding module (F), shared representation module (S), task-specific module (T), and loss criteria and training (L), as shown in Figure 2.

(I) Inputs: The MTMLF-QO model takes two types of inputs: (I.i) the data tables $T = \{T_1, T_2, \ldots, T_n\}$ in the DB and (I.ii) the query $Q = (T_Q, j_Q, f_Q)$ where $T_Q \subseteq T$ denotes the tables touched by Q, $j_Q = \{j(T_1, T_2), \ldots\}$ denotes the join predicates, and $f_Q = \{f(T_1), f(T_2), \ldots, f(T_n)\}$ denotes the filter predicates. We also provide Q's initial plan $\mathcal{P}$ with each node corresponding to a join or filter scan operation.

We modify the CardEst and CostEst tasks to let MTMLF-QO take the query $\mathcal{P}$ and estimate the cardinality and cost of the sub-plan rooted at each node of $\mathcal{P}$. All three tasks will be trained jointly.

(F) Featurization and encoding module: We try to extract the useful information from each data tuple and input queries, and embed them into vectors [Sun and Li 2019, Marcus et al. 2019] (shown in F.i). Specifically, we provide a value embedding for each unique column domain value in the DB to embed the tuples and the predicates $j(Q)$, $f(Q)$. We use a one-hot vector to represent each distinct table, column, and physical operation of a DB.

After the featurization, (F.ii) of this module will encode the data distribution of each table. For each single table T_i, this module deploys a transformer encoder [Vaswani et al. 2017] (Enc_i), which takes the filter predicate on this table $f(T_i)$ and outputs $E(f(T_i))$ representing the distribution of T_i after applying $f(T_i)$.

Thus far, we can embed each node N_i of the query plan $\mathcal{P}$ as a concatenation of the one-hot vector embedding of tables touched by N_i, the one-hot vector of operation type, and the embedding for predicate $j(N_i)$ or encoded $E(f(N_i))$. We denote the embedding of N_i as $E(N_i)$. At last, a *serializer* (F.iii) will convert the tree-structured plan into a vector $E(\mathcal{P}) = (E(N_1), E(N_2), \ldots)$ using the transformers' tree positional embedding techniques [Shiv and Quirk 2019].

(S) Shared representation module: After the previous module produces a sequence of embeddings $E(\mathcal{P})$ for the

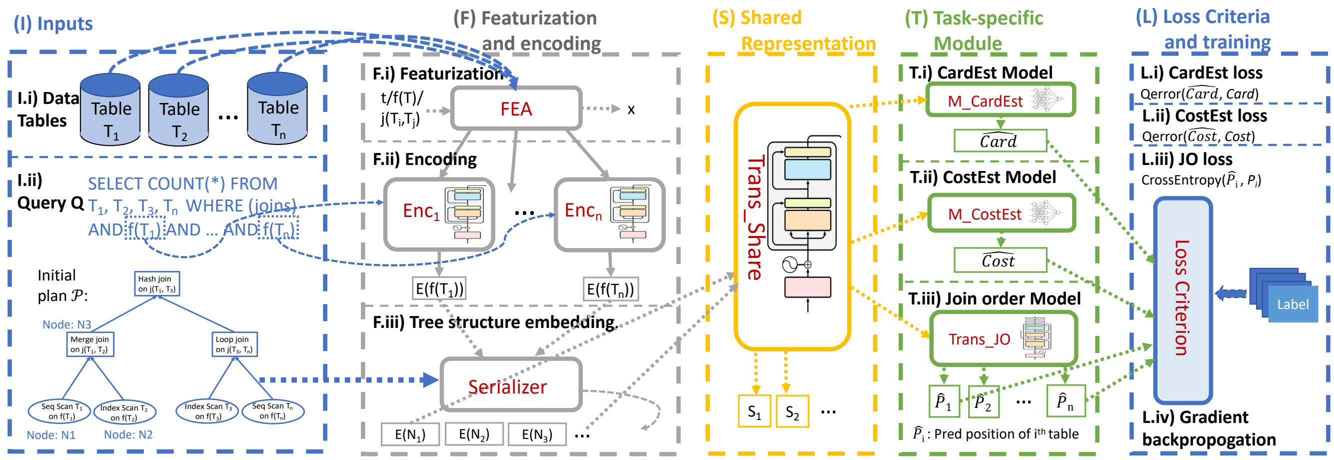

Figure 2: MTMLF-QO Model for a single DB

query, MTMLF-QO will model the interactions among elements $E(N_i)$ of $E(\mathcal{P})$ and generate a shared representation for the subsequent tasks. We use a transformer encoder *Trans_Share* to learn such interactions.

The input $E(\mathcal{P})$ to *Trans_Share* contains the information of single table distributions and join predicates. *Trans_Share* will construct the multi-table join distributions and understand the cost of different physical operations on specific single and join tables. The output $(S_1, S_2, \dots)$ of *Trans_Share* has the same length as the input, with one-to-one correspondence. For example, the S_i corresponding to $E(N_i)$ will represent the query $\mathcal{P}$'s sub-plan rooted at node N_i.

(T) Task-specific module: We use two multiple-layer perceptrons(MLPs), namely *M_CardEst* and *M_CostEst*, to directly extract the estimated cardinality $\widehat{Card}$ and cost $\widehat{Cost}$ from the shared representation (T.i and T.ii, respectively). However, extracting the optimal join order from this representation is much more complicated, because there exists an exponential number of possible join orders and a large amount of them might not be executable (e.g. there does not exist a join predicate between two tables).

As demonstrated in (T.iii), we formulate the JoinSel task into a sequence to sequence learning task (seq2seq) and use a transformer decoder [Vaswani et al. 2017] *Trans_JO* to generate the join order. For clarity of discussion, we focus on generating the left-deep join orders [Leis et al. 2015], which can be directly flattened into an ordered sequence of tables. Specifically, the *Trans_JO* takes as input the shared representation $(S_1, S_2, \dots)$, with each S_i representing a single or join table. At each timestamp t, *Trans_JO* will output a value $\hat{P}_t$ representing the probability of which table should be joined at the current timestamp. For a DB with n tables, $\hat{P}_t$ will be a multinoulli distribution vector of length n with the i-th entree corresponding to the probability that the table T_i is the next table to join with. Then, we design a novel decoding algorithm to decode a sequence of tables from the time sequence of $\hat{P}_t$ as the predicted join order, which is guaranteed to be legal and executable.

Please note that the *Trans_JO* can also generate bushy plans with our novel decoding algorithm based on beam-search [Boulanger-Lewandowski et al. 2013, Graves 2012]. We put the details of *Trans_JO* for bushy plans and the decoding algorithm in the technical report [Wu et al. 2021].

(L) Loss criteria and training: In order to train the models for CardEst and CostEst, we use the conventional Q-error loss [Kipf et al. 2019, Sun and Li 2019], i.e., the factor between the predicted and true cardinality or cost (L.i and L.ii): $L_{card} = max(\widehat{card}/card, card/\widehat{card})$.

For the JoinSel, which can be modeled as the seq2seq task, we use the cross-entropy loss function. Specifically, given a ground truth optimal left-deep join order $T'_1, T'_2, \dots, T'_m$ for a query Q touching m tables out of the total n tables, we can embed each T'_t into P_t, a one-hot vector of length n. At each time stamp t, the MTMLF-QO outputs a probability vector $\hat{P}_t$ and we can compute a cross entropy loss between $\hat{P}_t$ and P_t. We average the loss across all m timestamps and derive the loss of join order $L_{jo} = -(\sum_{t=1}^{m} P_t \cdot log(\hat{P}_t))/m$. This refers to the token-level loss function in NLP context [Graves 2012, Ranzato et al. 2015]. To empower MTMLF with the ability to effectively learn the optimal join orders, we design a novel sequence-level loss function to train the *Trans_JO*, whose details are provided in the technical report of this paper [Wu et al. 2021].

During the offline training phase of MTMLF-QO, all three tasks are trained jointly. The overall *loss criterion* is defined as the weighted combination of three loss functions for three tasks as defined in equation 1. The weights are hyper-parameters of the MTMLF-QO.

$$L_{QO} = w_{card} * L_{card} + w_{cost} * L_{cost} + w_{jo} * L_{jo} \qquad (1)$$

The gradient of this loss function will be backpropagated to update the parameters of the (S) and (T) modules only.

For the (F) module, each single table encoder Enc_i (F.ii) is trained separately with a CardEst task on a single table T_i. I.e. Enc_i learns the data distribution of T_i through predicting the cardinality of filter predicate $f(T_i)$. The details are put in the accompanied technical report [Wu et al. 2021].

Future research opportunities: The optimal join order for a query with a large number of tables is very expensive to obtain, limiting the MTF-QO's ability to extrapolate to very complex queries. A two-phase training can potentially alleviate this problem. I.e, an existing DBMS can be used to generate sub-optimal join orders to train a baseline MTF-QO, and then the precious data of the optimal join orders will be used to optimize this model.

Algorithm 1: Meta-learning Algorithm for MTMLF-QO

1: **Input**: n database $((D_1, Q_1), (D_2, Q_2), \ldots, (D_N, Q_N))$
2: Initialize empty set $Train\_Data$
3: **for** $i \in \{1, \ldots, n\}$ **do**
4: For each table T_j in D_i, train Enc_j (F.i and F.ii in Figure 2)
5: (F) module featurizes each query in Q_i, and derive $E(\mathcal{P})$
6: Add $(E(\mathcal{P}), Card, Cost, P_t)$ to $Train\_Data$
7: Shuffle $Train\_Data$
8: Train (S) and (T) modules with $Train\_Data$

3.3 Cross-DB meta learning for MTMLF-QO

In this section, we first propose a meta-learning algorithm (MLA) for MTMLF-QO and then conceptually reason about its feasibility.

Meta-learning algorithm: The details of MLA are shown in Algorithm 1. Assume that MTMLF-QO has access to n DBs, each with data tables D_i and executed query workload Q_i. The MLA aims at enabling MTMLF-QO to predict the cardinality, cost, and join order for all n DBs using a single model, and learning the *database-agnostic meta knowledge*. Thus, MLA empowers MTF-QO with the ability to transfer its learned knowledge to new DBs.

First, the data tables and queries in each DB will go through the featurization module of MTF-QO. Then, following the training procedure described earlier, we train the single table encoder Enc_j for each table T_j of each DB (line 4). Thus, the (F) module can embed each query $q \in Q_i$ with initial plan $\mathcal{P}$ (line 5) and add the embedding $E(\mathcal{P})$ and its corresponding cardinality, cost, and optimal join order to the training dataset (line 6). After all queries in all DBs have been added, MLA will shuffle the training dataset (line 7) and train the share representation (S) and task-specific modules (T) using the aforementioned loss criteria (line 8).

The returned MTMLF-QO trained by MLA would extrapolate to various DBs and produce accurate predictions on all of them. Thus, for each new DB, we can train the single table encoders (Enc_j in F.ii) and the "meta" MTMLF-QO model only needs to be fine-tuned on a small number of example queries. The encoders in (F.ii) only require query cardinalities on single tables and are efficient to train.

Conceptual reasoning of MLA: MLA pushes all data-specific information to the (F) module, which can be efficiently trained for a new BD. By shuffling the training dataset across different DBs, the MLA enforce subsequent modules of MTMLF-QO to learn the data-agnostic information, such as how the (S) module can derive the distribution on the join of multiple tables, and how the (T) module can use the shared representation to predict the cardinality, cost and join order. Without this training procedure, the (S) and (T) will merely map the embedded query to the target by "brute force" without truly understanding the semantics of the data distribution.

We provide a detailed example as follows of how the (S) and (T) modules can learn to construct the distribution on the join of multiple tables from single table distributions provided by the (F) module. Without MLA, the (S) module would require thousands of executed multi-table join queries to forcedly capture this information. Alternatively, the join tables probability distribution can be reconstructed from the single table distributions. For example, consider two tables A, B, and their join table $O = A \bowtie B$ on join predicate $A.id = B.id$. The probability of any filter predicate on O can be derived from the distributions on A and B only, as shown in Equation 2. By shuffling the training dataset across different DBs, MLA will compel the (S) module to learn this reconstruction process because otherwise a single (S) module can not extrapolate on different DBs.

$$P_O(f(A) \wedge f(B)) = \sum_{id \in D(A.id)} P_A(f(A) \wedge A.id = id) * P_B(f(B) \wedge B.id = id) \tag{2}$$

Table 1: Q-errors on the JOB workload.

Method	Cardinality			Cost		
	median	max	mean	median	max	mean
PostgreSQL	184.00	670,000	10,416	4.90	4920	105.00
Tree-LSTM	8.78	696.29	36.83	4.00	290.35	15.01
MTMLF-QO	**4.48**	**614.45**	**28.69**	**2.10**	**37.54**	**4.20**
MTMLF-CardEst	5.12	804.48	36.66	\	\	\
MTMLF-CostEst	\	\	\	2.06	61.41	4.69

Table 2: Execution time with different join orders.

JoinOrder	Total Time	Overall Improvement Ratio
PostgreSQL	1143.2 min	\
Optimal	209.1 min	81.7%
MTMLF-QO	**318.3 min**	**72.2%**
MTMLF-JoinSel	450.4 min	60.6%

4. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of MTMLF-QO for a single DB on CardEst, CostEst, and JoinSel tasks and the effectiveness of multi-task joint training. Then, we show the cross-DB "transferability" of MTMLF-QO trained on multiple DBs via MLA to a new DBs.

4.1 Experiments on single DB

We use the JOB benchmark of 113 queries joining dozens of tables and having the complex "LIKE" predicates on the IMDB dataset containing 21 tables with skewed distribution and strong attribute correlation [Leis et al. 2015]. Following the prior work [Sun and Li 2019], we generate 150K SQL queries similar to the JOB queries as the training data. Then, we execute these queries in PostgreSQL [Group 2018] to derive the query plans and their true cardinalities and costs. For the JoinSel task, we generate the optimal join order using the ECQO program [Trummer 2019]. Since deriving optimal join order has exponential time complexity, we can only afford to execute this program for 20K queries out of the 150K, which touches no more than 8 tables.

Hyperparameters of MTMLF-QO: We use a transformer with 3 blocks and 4 headers for each Enc_i, the *Trans_Share* and the *Trans_JO*. We use two-layer MLPs for *M_CardEst* and *M_CostEst*. The weights w_{card}, w_{cost}, and w_{jo} are all set to 1. The Adam optimizer [Kingma and Ba 2014] with 10^{-4} learning rate is used to optimize the model. All experiments are conducted on a CentOS Server with an Intel Xeon Platinum 8163 2.50GHz 64-core CPU, 376GB DDR4 main memory, and 1TB SSD and GeForce RTX 2080 Ti GPU.

Table 3: Execution time with different join orders.

JoinOrder	Total Time	Overall Improvement Ratio
PostgreSQL	393.9 min	\
MTMLF-QO (MLA)	**234.1 min**	**40.6%**
MTMLF-QO (single)	219.5 min	44.3%

Performance on CardEst and CostEst: In order to show the effectiveness of our MTMLF-QO model on CardEst and CostEst tasks, we compare it with a traditional DBMS PostgreSQL [Group 2018], and the previous SOTA method Tree-LSTM [Sun and Li 2019] on the JOB benchmark. Please note that we can not compare MTMLF-QO with other data-driven SOTA methods [Zhu et al. 2021, Wu and Shaikhha 2020, Hilprecht et al. 2020] because they can not support "Like" predicates on strings [Han et al. 2021].

We take 90% of generated 150K queries as the training dataset, 10% as the validation set for hyper-parameter tuning, and JOB queries as the test set. We use q-error as the metric to evaluate cardinality and cost estimation. As shown in Table 1, our MTMLF-QO significantly outperforms the traditional DBMS and the previous SOTA Tree-LSTM on both CardEst and CostEst tasks.

Performance on JoinSel: To evaluate the quality of the join order generated by MTMLF-QO, we use 85% of the 20K queries to train, 10% of the queries to find the hyper-parameter, and the rest 5% as the test set to predict the optimal join orders. Note that, we refrain from testing on the original JOB queries because MTMLF-QO only has access to queries joining no more than 8 tables.

We compare the quality of the join order generated by MTMLF-QO against with two baselines: the original PostgreSQL's query optimizer and the optimal join order produced by ECQO. Table 2 shows the results of query execution time using different join orders, where "total time" is the total running time of all $1,000$ testing queries, and "overall improvement ratio" refers to the improvement over the PostgreSQL divided by the PostgreSQL total time.

Based on this table, we can see that the learned join order of MTMLF-QO can significantly outperform the PostgreSQL baseline. In addition, for more than 70% of the $1,000$ testing queries, MTMLF-QO can output the optimal join order. These results indicate that MTMLF-QO can be a very effective learned query optimizer of PostgreSQL. We left the comparison of MTMLF-QO with other SOTA join order selection methods [Marcus et al. 2019, Marcus and Papaemmanouil 2018, Guo and Daudjee 2020, Yu et al. 2020] as future work.

Benefits of multi-task joint training: In order to demonstrate the benefits of multi-task joint training of MTMLF-QO, we conduct an ablation experiment to separately train the MTMLF-QO model for CardEst (MTMLF-CardEst), CostEst (MTMLF-CostEst), and JoinSel (MTMLF-JoinSel). By Table 1 and Table 2, the performance of MTMLF-JoinSel is much worse than the original MTMLF-QO, and MTMLF-CardEst and MTMLF-CostEst are also slightly worse than MTMLF-QO. This suggests that the joint training of CardEst, CostEst, and JoinSel tasks is indeed more effective than the separate training.

4.2 Experiments on cross-DB transferrability

Artificial DBs: Since there exists a very limited number of open-source real-world DBs, we generate artificial DBs to verify the cross-DB transferability. Specifically, we design a data generation pipeline to produce 11 DBs, each containing $6-11$ tables with a varied number of attributes and very different distributions. The details of this pipeline are provided in the technical report [Wu et al. 2021].

Experiment procedures: We first generate 11 artificial DBs $\{\mathcal{D}_1, \ldots, \mathcal{D}_{11}\}$. For each DB $\mathcal{D}_i$, we create a workload W_i of $20K$ join queries and execute the ECQO program [Trummer 2019] to derive its optimal join order.

The hyper-parameters of MTMLF are the same as described in Section 4.1 of the main paper. The training of the MTMLF follows the MLA procedure. Specifically, we first generate some single-table queries for each table within each DB $\mathcal{D}_i$. Using these queries, we train a featurization module F_i for every DB to capture all the dataset-specific knowledge such as the single table distributions. The procedure of training each F_i is very efficient since the single table query can be efficiently executed in parallel or using AQP techniques. Then, we use 10 DBs $\{\mathcal{D}_1, \ldots, \mathcal{D}_{10}\}$ as the training data and learn the (S) shared representation and (T) task-specific modules for the MTMLF via MLA described in Section 3.3. These two modules are able to output effective join orders for all 10 DBs. Thus, it must have captured the dataset-agnostic knowledge that can be transferred to a new DB.

We use $\mathcal{D}_{11}$ as testing data to verify the transferability of MTMLF. Specifically, we connect the learned F_{11} module containing all dataset-specific information of $\mathcal{D}_{11}$ with the pre-trained (S) and (T) modules. Then, we use this MTMLF model to generate the join order of queries on this DB and execute these join orders in PostgreSQL.

Effectiveness of MTMLF-QO's meta-learning: From Table 3, we observe that the MTMLF-QO trained via MLA can generate join orders that are 40% faster than the ones produced by PostgreSQL baseline on a brand new DB. As a controlled study, we directly train an MTMLF-QO on this test DB $\mathcal{D}_{11}$ from scratch (MTMLF-QO single), which is only slightly better than MTMLF-QO trained via MLA. These results suggest that MTMLF-QO has the capacity to distill cross-DB meta-knowledge that is transferrable to new DBs.

5. CONCLUSIONS

In this paper, we present the MTMLF, which can condense an effective shared representation to mutually benefit various tasks in DBMS and distill the "meta-knowledge" beneficial to all DBs. We also demonstrate with a very promising case study on query optimization that future research along this direction can be fruitful.

Next, we list two concrete future research opportunities. First, inspired by MTMLF-QO, other DBMS tasks can also be incorporated into the MTMLF framework. Second, a cloud DB service can greatly facilitate the *pre-train fine-tune* paradigm of MTMLF. This setting motivates the research community to design a federated learning algorithm to protect the DB users' data privacy and simultaneously ensure effective training of MTMLF.

6. REFERENCES

[Basu et al. 2016] Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. 2016. Regularized cost-model oblivious database tuning with reinforcement learning. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII*. Springer, 96–132.

[Boulanger-Lewandowski et al. 2013] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. 2013. Audio Chord Recognition with Recurrent Neural Networks. In *ISMIR*. Citeseer, 335–340.

[Brown et al. 2020] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[Deng et al. 2020] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: Table understanding through representation learning. *arXiv preprint arXiv:2006.14806* (2020).

[Devlin et al. 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[Ding et al. 2020a] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020a. ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 969–984.

[Ding et al. 2020b] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020b. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *arXiv preprint arXiv:2006.13282* (2020).

[Graves 2012] Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711* (2012).

[Group 2018] The PostgreSQL Global Development Group. 2018. *Documentation PostgreSQL 10.3*.

[Guo and Daudjee 2020] Runsheng Benson Guo and Khuzaima Daudjee. 2020. Research challenges in deep reinforcement learning-based join query optimization. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–6.

[Han et al. 2021] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *arXiv preprint arXiv:2109.05877* (2021).

[Herzig et al. 2020] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349* (2020).

[Hilprecht et al. 2020] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kersting Kristian, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *PVLDB* (2020).

[Kingma and Ba 2014] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[Kipf et al. 2019] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating correlated joins with deep learning. *CIDR* (2019).

[Konečnỳ et al. 2015] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).

[Kraska et al. 2018] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. 489–504.

[Leis et al. 2015] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow* 9 (2015), 204–215.

[Li et al. 2019] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[Ma et al. 2020] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active learning for ML enhanced database systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 175–191.

[Marcus et al. 2019] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A learned query optimizer. *PVLDB* (2019).

[Marcus and Papaemmanouil [n.d.]] Ryan Marcus and Olga Papaemmanouil. [n.d.]. Wisedb: A learning-based workload management advisor for cloud databases. *PVLDB* ([n. d.]), 780–791.

[Marcus and Papaemmanouil 2018] Ryan Marcus and Olga Papaemmanouil. 2018. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–4.

[Marcus and Papaemmanouil 2019] Ryan Marcus and Olga Papaemmanouil. 2019. Plan-structured deep neural network models for query performance prediction. *arXiv preprint arXiv:1902.00132* (2019).

[Nathan et al. 2020] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 985–1000.

[Negi et al. 2021] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *arXiv preprint arXiv:2101.04964* (2021).

[Ortiz et al. 2018] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S Sathiya Keerthi. 2018. Learning state representations for query optimization with deep reinforcement learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*. 1–4.

[Ranzato et al. 2015] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732* (2015).

[Sheng et al. 2019] Yangjun Sheng, Anthony Tomasic, Tieying Zhang, and Andrew Pavlo. 2019. Scheduling OLTP transactions via learned abort prediction. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–8.

[Shiv and Quirk 2019] Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. *Advances in Neural Information Processing Systems* 32 (2019), 12081–12091.

[Siddiqui et al. 2020] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost models for big data query processing: Learning, retrofitting, and our findings. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 99–113.

[Sun and Li 2019] Ji Sun and Guoliang Li. 2019. An end-to-end learning-based cost estimator. *arXiv preprint arXiv:1906.02560* (2019).

[Tang et al. 2020] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2020. Relational Pretrained Transformers towards Democratizing Data Preparation [Vision]. *arXiv preprint arXiv:2012.02469* (2020).

[Trummer 2019] Immanuel Trummer. 2019. Exact cardinality query optimization with bounded execution cost. In *Proceedings of the 2019 International Conference on Management of Data*. 2–17.

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

[Wu and Shaikhha 2020] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *arXiv preprint arXiv:2012.14743* (2020).

[Wu et al. 2021] Ziniu Wu, Peilun Yang, Pei Yu, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2021. A Unified Transferable Model for ML-Enhanced DBMS. *arXiv preprint arXiv:2105.02418* (2021).

[Yang et al. 2020] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *arxiv* (2020).

[Yang et al. 2019] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep unsupervised cardinality estimation. *Proceedings of the VLDB Endowment* (2019).

[Yu et al. 2020] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement learning with tree-lstm for join order selection. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1297–1308.

[Zhang et al. 2019] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.

[Zhou et al. 2020] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[Zhu et al. 2021] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *VLDB* (2021).