The Case for Multi-Task Zero-Shot Learning for Databases

Johannes Wehrstein
Technical University of Darmstadt

Benjamin Hilpecht
Technical University of Darmstadt

Benjamin Olt
Technical University of Darmstadt

Manisha Luthra
Technical University of Darmstadt

Carsten Binnig
Technical University of Darmstadt & DFKI

ABSTRACT

Recently, machine learning has successfully been applied to many database problems such as query optimization, physical design tuning, or cardinality estimation. However, the predominant paradigm to design such learned database components is workload-driven learning, where a representative workload has to be executed on the database to gather training data. This costly procedure has to be repeated for every new database a model should be trained on. Hence, recently it was suggested to train zero-shot cost models that are pretrained once and can generalize to unseen databases out-of-the-box. While the results for the task of cost estimation are promising, it is unclear how to generalize this approach to additional tasks beyond query latency prediction. Hence, in this paper, we propose several directions to generalize zero-shot cost models to other tasks and validate our approaches in two case studies.

AIDB Workshop Reference Format:
Johannes Wehrstein, Benjamin Hilpecht, Benjamin Olt, Manisha Luthra, and Carsten Binnig. The Case for Multi-Task Zero-Shot Learning for Databases. *AIDB* 2022.

1 INTRODUCTION

In recent years, learned database components have gained traction in both research and (increasingly) also in commercial systems where the main idea is to replace manually designed components with a machine learning (ML) model. The promise is that previously hard to solve problems such as query optimization, which incur immense engineering efforts, can be solved automatically and more accurately using ML. In particular, there have been efforts to not only using ML for query optimization [24, 26, 33] including cardinality and cost estimation [9–11, 13, 16, 27, 30, 34], but also other tasks in DBMSs such as query scheduling [29] or learned physical designs [12, 17, 21].

The predominant approach to design learned database components today is workload-driven learning where the idea is to run a representative workload on the database and use this as training data for the ML model [13, 16]. For instance, in order to train a learned query optimizer, one has to execute thousands of queries and use the physical plans and their runtime as training data. Once the model is trained, the deployed model can then be used to suggest efficient physical execution plans. However, while workload-driven learning has shown promising results for many of the aforementioned tasks, the execution of training workloads is costly especially for larger databases since thousands of queries are required for training the model. Even worse, we have to repeat this process for every new database (i.e., a dataset with a given schema and workload) at hand since the model architectures used for workload-driven learning can only be trained and used on a single (fixed) database due to the representation they use. In general, there is no straightforward way to use the same model across databases.

In contrast, recently proposed zero-shot cost models [10] tackle this problem by using a new transferable representation for data and queries. Consequently, zero-shot cost models are able to provide cost estimates (i.e., query latency predictions) for unseen databases out-of-the-box. The idea of such zero-shot models is to (once) pretrain a cost model over workload traces of different databases and thus allow the model to generalize to a new database without additional training data. This alleviates the previously mentioned high costs incurred for workload-driven models for every new database. However, currently, zero-shot cost models are limited to the task of cost estimation and more precisely to query latency prediction. While this is an important task, it is unclear how to extend the proposed models to other database tasks such as predicting query optimization, query scheduling, or physical design tuning.

Contributions. The main goal of this paper is to present different avenues on how to generalize the idea of zero-shot models for database components to a broader set of tasks beyond query cost estimation. This enables learned database components that are cheap to deploy for a new database (like zero-shot cost models) but that also support a much broader set of tasks, which can only be tackled today using workload-driven learning. Overall, we envision that this paper can thus open up the way to provide zero-shot models for a rich set of different database components ranging from zero-shot learned storage layouts (e.g., learned physical designs) over zero-shot query optimization to zero-shot query scheduling and thus enable learned database systems that can be instantiated on a new database and workload with no or minimal training overhead.

In order to enable zero-shot models for a much broader set of database components, as a core contribution in this paper, we explore three main directions: *(i)* fine-tune zero-shot cost models, *(ii)* combination with optimization approaches, and *(iii)* end-to-end zero-shot models. As we discuss later in this paper, all three directions come with particular advantages which make them a good fit for a particular DBMS component. For instance, fine-tuning zero-shot models can build on the existing model architectures for zero-shot cost models. Hence, this direction is the least involved method since no new model architectures need to be developed and only new training data for fine-tuning is needed. However, fine-tuning only enables zero-shot models for a limited set of tasks. The other two directions, while more involved, can in contrast support a much broader set of components and tasks as we discuss in detail in the paper.

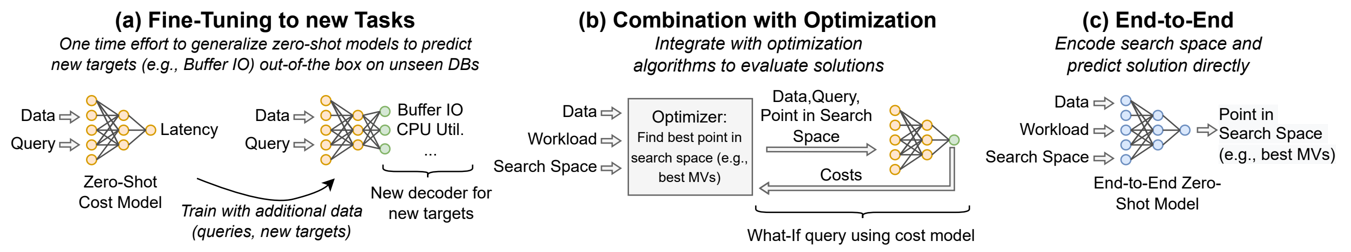

(a) Fine-Tuning to new Tasks
One time effort to generalize zero-shot models to predict new targets (e.g., Buffer IO) out-of-the box on unseen DBs
Data
Query
Latency
Zero-Shot Cost Model
Train with additional data (queries, new targets)
Data
Query
Buffer IO
CPU Util.
...
New decoder for new targets
(b) Combination with Optimization
Integrate with optimization algorithms to evaluate solutions
Data
Workload
Search Space
Optimizer: Find best point in search space (e.g., best MVs)
Data,Query, Point in Search Space
Costs
What-If query using cost model
(c) End-to-End
Encode search space and predict solution directly
Data
Workload
Search Space
Point in Search Space (e.g., best MVs)
End-to-End Zero-Shot Model

Figure 1: Proposed Directions for Generalizing Zero-Shot Learning to a broad set of Tasks. As a first direction, we suggest to fine-tune zero-shot cost models to a new task (a), where we train existing zero-shot cost models further on training data capturing different targets (e.g., buffer I/O) which however only allows existing zero-shot cost models as suggested in [10] to generalize to simple supervised regression tasks on queries. Hence, as a second direction, we suggest including zero-shot cost models in existing optimization approaches (b) that aim to find a suitable solution in a search space (e.g., a set of materialized view (MV) candidates). Here, the zero-shot cost models need to be extended by a so-called what-if mode which allows the model to evaluate the costs of certain points in the design space (e.g., the benefit of a set of MV candidates). Finally, as the last direction in (c), we suggest end-to-end zero-shot models that directly predict a suitable point in the search space where the challenge is to encode the entire search space, which has the potential to find better solutions in a shorter time.

To validate our ideas, we present initial results of two case studies to evaluate these directions for two novel tasks that were previously not supported by zero-shot cost models: *(i)* we show how we can fine-tune zero-shot cost models to support the prediction of query metrics other than query latency such as buffer I/O as well as CPU utilization, and *(ii)* we also discuss how we can leverage zero-shot cost models to support more complex tasks such as learned storage layouts and in particular focus on the task of learned materialized view (MV) selection.

Outline. In Section 2, we first provide the necessary background on zero-shot learning for databases before we give an overview of the three main directions to generalize zero-shot models to additional tasks in Section 3. Afterwards, we then discuss each direction in detail in Section 4 to Section 6. Finally, we give an overview about related work in Section 7 before we conclude in Section 8.

2 ZERO-SHOT LEARNING FOR DATABASES

In this section, we provide necessary background on zero-shot models for learned databases [9]. The goal of zero-shot learning is to learn models for database tasks such as cost estimation [10] that can generalize to unseen databases (new schemas, datasets and workloads) out-of-the-box without requiring additional training queries.

In contrast, state-of-the-art workload-driven models for tasks such as cardinality estimation [16, 30] or query optimization [27, 34] require ten thousands of query executions for every unseen database as training data since the models and training data are tied to a single database. Depending on the size of the database, gathering the training data can be a significant effort: for a database size of 1 TB it would require more than six months [31] to gather a data set of ten thousand runtime measurements. This is especially a burden for cloud database vendors such as Snowflake or Redshift, where it would be necessary to incur these costs for each different customer and their databases.

The main idea of zero-shot models to avoid these high costs is to pre-train a model on a variety of different databases and workloads as shown in Figure 2. Afterwards, the model can be used on an unseen database out-of-the-box, i.e., without further training. One

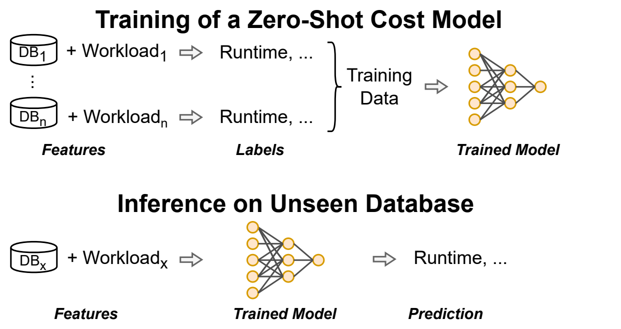


Figure 2: Overview of training and inference of zero-shot models for database tasks. The training on a variety of datasets is a one-time-effort which allows the usage of the trained model for any unseen database out-of-the-box [10].

might now argue that the effort to obtain such workload traces on a variety of different databases might be a substantial effort. However, especially in the cloud, often logs of query executions are already kept and could thus readily be used for training zero-shot models. More importantly, gathering training data and training zero-shot models is a one-time-effort since they can generalize to unseen databases and thus the costs quickly amortize.

While zero-shot models address the high costs of training data collection for learned database components, it was only shown how such models can be constructed for cost estimation [8, 9] so far. Hence, in the next section, we introduce three directions of how zero-shot models can be generalized to new tasks.

3 MULTI-TASK ZERO-SHOT LEARNING

The main goal of this paper is to explore directions on how to obtain zero-shot models that can solve a broad set of tasks in the same way as zero-shot cost models [10] predict costs, i.e., the resulting models should be able to solve the task on an unseen database without additional training data. Intuitively, zero-shot cost models enable this for cost estimation by using a novel transferable representation of queries and data. While this enables to predict costs on unseen databases out-of-the-box, it is not trivial how this idea can be extended to support other tasks such as selecting a set of MVs for a particular workload.

3.1 Generalizing Zero-Shot Models

Hence, as a core contribution in this paper, we propose three main directions as depicted in Figure 1 to generalize zero-shot models: *(i)* fine-tuning to new tasks, *(ii)* combination with optimization algorithms, and *(iii)* end-to-end zero-shot models. In the following, we provide a brief overview of each direction.

Fine-tuning to new Tasks. In general, fine-tuning is a paradigm in ML, where a trained model for one task is used as a starting point to be further trained on a related task for which less data is available. In the context of zero-shot learning, we can use a pretrained zero-shot cost model and fine-tune it with additional training data for a different task such as predicting other query metrics, e.g., buffer I/O instead of the query latency (cf. Figure 1 (a)). Since the model has (prior to fine-tuning) already internalized the general characteristics of query operators to predict query latency, it can more easily learn to predict other query-related metrics. Importantly, fine-tuning a zero-shot model to a new metric is again only a one-time-effort, i.e., after fine-tuning, the resulting zero-shot cost models can predict the new metric on unseen databases without additional training just like zero-shot cost models can predict query latencies for unseen databases out-of-the-box. While fine-tuning can make use of the same query and data representation as the original zero-shot cost models, it only works for generalizing zero-shot cost models to other regression tasks; i.e., for predicting new query metrics such as buffer I/O instead of query latency.

Combination with Optimization Procedures. An interesting observation is that many of the core and thus performance-critical DBMS components solve optimization problems that combine a search procedure with a cost model. For example, in query optimizers, search procedures such as dynamic programming are applied to enumerate different possible query plans that are then evaluated by a cost model to pick a plan with minimal cost. Moreover, many other problems such as selecting suitable MVs for a workload can also be framed as an optimization problem [1], where a set of MVs should be selected such that the overall runtime of a given workload is minimized.

Hence, as a second direction shown in Figure 1 (b), we leverage existing optimization approaches and use them in combination with zero-shot cost models to evaluate certain points in the search space and pick the one which minimizes (or maximizes) a cost function. For example, to find a set of MV candidates that minimize the workload runtime, we could use an optimization procedure that can enumerate MV candidates and use zero-shot cost models to predict the expected runtime of a workload for the given set of MV candidates. However, to enable this, zero-shot cost models need to be extended to support a so-called *what-if mode.* In particular, we have to encode not only the data and query as input to a zero-shot model but also the point in the search space that needs to be evaluated. In the case of MVs, for instance, the set of materialized view candidates for which the runtime of queries needs to be encoded is also an input to the zero-shot model. This allows us to use a model and ask for the runtime *what-if* a certain set of MVs are selected.

End-to-End. Finally, as the last direction, we propose an *end-to-end* approach, where our goal are zero-shot models that directly solve the underlying problem. As depicted in Figure 1 (c), we thus aim to design zero-shot models that take the definition of the search space directly as input and output a point in the search space that is supposed to be optimal without involving an optimization procedure explicitly. For instance, for MV selection, the zero-shot model would thus directly suggest a suitable set of MVs as output instead of using a search procedure that enumerates different candidates.

Overall, using such an end-to-end approach has several advantages: first, search procedures in DBMS components typically rely heavily on heuristics or make other simplifying assumptions. For instance, for selecting MVs, the search space is often aggressively pruned as we discuss later leading to non-optimal solutions. Second, the time to find a solution for a new database and workload might be reduced since no costly optimization problem has to be solved. Therefore, we argue that end-to-end zero-shot models have the potential to find better solutions in a shorter time compared to a combination of a search procedure with a zero-shot model as discussed in the direction before.

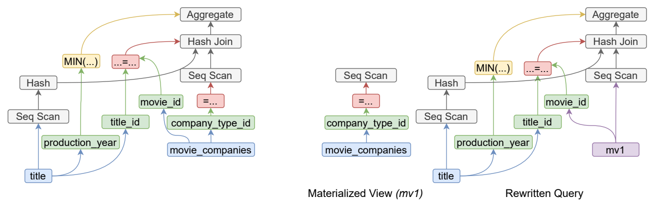
3.2 Discussion of Directions

All three before-mentioned directions come with their own advantages which make them a good fit for specific tasks. In particular, fine-tuning enables the adaption to new tasks with only little training data and minor modifications of the original zero-shot cost model. However, it is limited to regression tasks, e.g., predicting other resource metrics of queries such as buffer I/O.

In contrast, the direction where we combine zero-shot models with optimization algorithms can support a broader set of tasks. Moreover, a huge advantage of this direction is that it still offers some debuggability to database administrators of how a learned model came up with a decision (by inspecting evaluated candidates and their cost predictions). However, this direction has also several limitations such as the time to search the space of possible solutions and suggest an ideal candidate (e.g., a set of MVs). In addition, since optimization procedures often make use of heuristics (e.g., to prune the search space), solutions that are found by this direction might still not be optimal in many cases.

Consequently, this motivates the end-to-end approach where the model is directly trained on the end-to-end metric to also learn to navigate the search space and select an ideal candidate based on a given metric. However, this approach is also very challenging depending on the concrete task since the entire search space has to be encoded as model input in a way that it again generalizes across databases (e.g., the set of possible materialized views given a query workload and a database).

Finally, we want to note that we can also imagine that for some tasks it might be beneficial to combine the different approaches. For instance, in query scheduling, we might want to fine-tune a model to predict the memory consumption and afterwards embed this model in an optimization algorithm that makes the actual scheduling decision. In the next section, we detail each of the proposed directions.



(a) Encoding of a Query Plan (b) Rewriting Query Plan to MV

Figure 3: Queries are encoded as directed acyclic graphs in the zero-shot cost model containing different nodes types for tables (blue), attributes (green), operators (grey) and edges connecting them as shown in (a). For the case study of MV selection (cf. Section 5), the graph representation includes the MVs as new node type by replacing the respective subplan as shown in (b).

4 FINE-TUNING TO NEW TASKS

In this section, we detail the first approach of fine-tuning zero-shot cost models. We first explain the general methodology of fine-tuning before we explain a concrete example where we apply the methodology and provide the respective experiments.

4.1 Idea and Methodology

The main purpose of fine-tuning is to generalize a zero-shot cost model to other (related) tasks. For instance, a zero-shot cost model that was originally trained to predict query latency can be fine-tuned for another metric such as CPU utilization of the query execution. The advantage of fine-tuning is that it needs much less training data in contrast to training a zero-shot model for the new task from scratch. Another important point to note is that after fine-tuning, the zero-shot model can be used to predict the new metric out-of-the box for unseen databases without any further training. To realize an efficient fine-tuning of a zero-shot cost model with only a few training samples, the key idea is that we replace the decoder of an existing zero-shot cost model with a new decoder, which can then be trained with only minimal overhead (cf. Figure 1 (a)) for the new task. In the following, we discuss in more detail how the fine-tuning of a zero-shot cost model can be realized.

To better understand this approach, let us first revisit the anatomy of zero-shot cost models for query latency prediction [9, 10]. At the core of zero-shot models is a new graph-based representation of query plans. For query latency prediction, the query plan and data that are needed for the prediction are represented within the graph structure with different node types for operators, attributes, tables, and predicate information. An example of our graph representation of a query plan is shown in Figure 3 (a). In order to learn latency prediction across databases, the zero-shot cost model is trained with a set of so-called *transferable features* that allow generalization across databases; e.g., to predict query latency for different databases.

An important aspect of transferable features is that they can be derived from *any* database, which in turn enables the model to generalize to unseen databases. For instance, data characteristics such as table width, number of rows as well as input and output cardinalities of operators in a query are examples of such transferable features. Based on these features an embedding is computed per node in the graph representation (using a node-type specific MLP), and afterwards a message passing scheme is applied, which propagates the information through the graph (i.e., from leaves to the root node of the query plan). Finally, a multi-layer perceptron (MLP) is used as a decoder on top of the root node to derive the query latency estimate.

In the following, we now discuss how such a zero-shot model that was originally trained for query latency prediction can be fine-tuned for another task (e.g., predicting CPU utilization) of query plans across databases. Overall, three main modifications are required to the zero-shot cost model for fine-tuning as we explain below:

(1) Model Initialization. As a first step, we replace the decoder of a given zero-shot cost model to support different metrics as we can see in Figure 1(a). For deriving the model architecture for the new task, in addition, we take the remaining pre-trained weights of the zero-shot cost model (i.e., the weights of the nodes in the graph) for the encoder as initialization, whereas the weights of the decoder are initialized randomly[1]. The main idea behind this initialization is that it already covers query characteristics (in the context of cost estimation), which is the reason why we need less training data for the fine-tuning task.

(2) Fine-tuned Training. In order to fine-tune the initialized zero-shot model with the new decoder, we retrain the model with additional data for which we want to refine the model. For instance, to support additional cost metrics such as buffer I/O or CPU utilization, we re-train the model with these additional metrics. In this fine-tuning procedure, we retrain all model weights (i.e., the ones of the graph model as well as the weights of the new decoder). We also tried out other training procedures for fine-tuning such as only training the decoder weights while freezing the weights of the graph nodes or using other techniques such as refining parameters with regularization [5]. However, these techniques did not improve the fine-tuning compared to the approach we use (i.e., fine-tune all parameters without regularization). It is important to note, however, that the number of queries required for fine-tuning are way less than the number of queries required for training a zero-shot cost model from scratch. In our experiments in the next section, we will show this effect in more detail.

(3) Inference with the Fine-tuned Model. At inference time, the model with the new decoder can be used to predict the additional cost metrics as explained above. Here, the main procedure for the cost estimation remains similar to zero-shot cost models, i.e., *(i)* compute the hidden state for every node of the graph structure, *(ii)* combine information from the different graph nodes using message passing and *(iii)* predict the additional cost metrics using the final MLP stage.

4.2 Case Study and Results

In order to show the efficiency of fine-tuning, we conduct a case study where we fine-tune the zero-shot cost estimation model [9]

[1]More precisely, we actually only randomly initialize the last layers of the decoder MLP, whereas the first layers take the weights of the pre-trained zero-shot cost model for query latency.

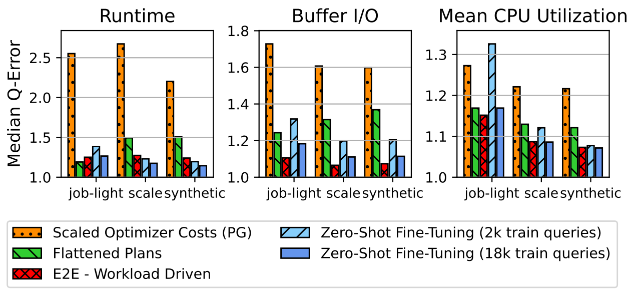


Figure 4: Fine-tuned zero-shot cost models can robustly predict additional cost metrics such as Buffer I/O as well as CPU Utilization with high accuracy on unseen databases similar to zero-shot models. The graph shows median q-error in comparison to the flattened plan approach [6, 15], end-to-end workload-driven [30] and scaled optimizer costs for the metrics: runtime, buffer I/O and mean CPU utilization, respectively.

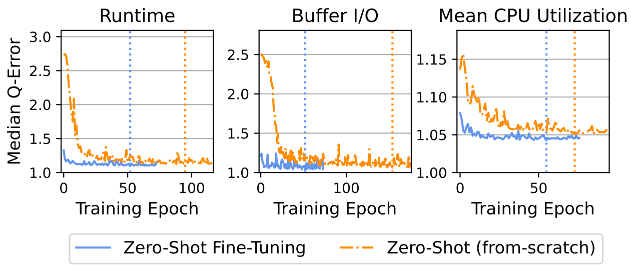
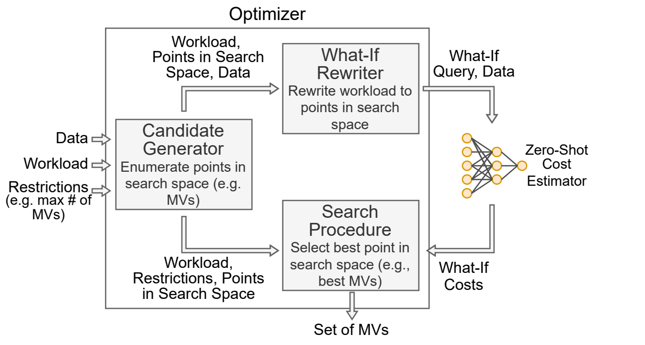


Figure 5: Fine-tuned model can from the beginning of training, predict cost metrics with very high accuracy in comparison to the model that needs to be trained from scratch. Thus, the convergence time (shown as vertical lines) is reduced for the fine-tuned model compared to its counterpart.

for several additional cost metrics (besides query latency) such as buffer I/O and CPU utilization. We present the estimated errors of our fine-tuned model that is re-trained with different training data sizes (see Figure 4) as well as the learning curves of fine-tuning vs. training a zero-shot model for the new tasks from scratch (see Figure 5).

Setup and Benchmarks. As a test workload, we use the IMDB data with three different benchmarks JOB-light, scale and synthetic [16]. We compare our fine-tuning approach to three related approaches: the first approach is based on Postgres cost estimates (called scaled optimizer). The other two baselines are approaches based on workload-driven learning, which need to run a training workload per database: flattened plans [6, 15] and an approach that uses a graph representation of query plans [30] but without transferable features (called e2e).

Note that the scaled optimizer from Postgres was used already in [9] as a baseline and applies a simple linear model to predict the cost metric (e.g., CPU utilization) based on the Postgres estimates. Moreover, all workload-driven approaches are trained on the target database (i.e. IMDB) with a workload of 50k queries. Different from those approaches, our fine-tuned zero-shot cost model has not seen any training queries on IMDB. Instead, the fine-tuned zero shot



Optimizer
Workload, Points in Search Space, Data
What-If Rewriter
Rewrite workload to points in search space
What-If Query, Data
Data
Workload
Restrictions (e.g. max # of MVs)
Candidate Generator
Enumerate points in search space (e.g. MVs)
Zero-Shot Cost Estimator
Search Procedure
Select best point in search space (e.g., best MVs)
Workload, Restrictions, Points in Search Space
What-If Costs
Set of MVs

Figure 6: Combining zero-shot cost estimator with optimization procedure, e.g., for MV selection. The optimizer takes as an input data, workload and restrictions, e.g., number of views and enumerates over the query plans with MV, rewrites queries in what-if mode, which are used by the zero-shot model to estimate costs for the given MV candidates.

model uses a much smaller number of training queries on several databases, which allows the zero-shot model to generalize to new unseen databases (such as IMDB) out-of-the box. For comparison, we use Q-error as it is a standard metric to compare ML-based cost models, measured as $q(c, \hat{c}) = max\left(\frac{c}{\hat{c}}, \frac{\hat{c}}{c}\right)$, with $q \geq 1$ [28].

Initial Results. In Figure 4, we show the performance of the three different baselines in comparison to our fine-tuned zero-shot cost model. The main observation here is that even though the fine-tuned model has not seen the IMDB dataset in training, it predicts the cost metrics with very high accuracy (the median Q-error is below 1.5 for most of the benchmarks). In fact, more interestingly, the fine-tuned model that has seen as low as 2k queries for the additional cost metrics already has a very low Q-error (<1.5), and we see an improvement in the accuracy when the model is fine-tuned with even more queries (18k) leading to a Q-error of < 1.3.

Even more importantly, besides requiring less training data, fine-tuning also significantly reduces the training time and provides higher accuracy simultaneously compared to training a zero-shot model from scratch. In Figure 5, we depict the prediction errors over the course of training of both a fine-tuned model (taking a zero-shot model predicting different metrics as a starting point) as well as zero-shot model that is trained from scratch. As we can see, right in the beginning, the fine-tuned model can predict the new metric more accurately and also converges much faster and thus the overall training time is reduced.

5 COMBINATION WITH OPTIMIZATION

While fine-tuning a zero-shot model enables the adaption to new tasks with only minor modifications of the original zero-shot cost model, it is limited to supervised learning tasks on queries, e.g., predicting other resource metrics of queries such as buffer I/O. In the following, we thus describe the second direction of combining zero-shot cost models with optimization procedures in more detail.

5.1 Idea and Methodology

The main purpose of this direction is to enable zero-shot cost models to solve database tasks that involve navigating complex typically non-differentiable search spaces. The advantage of this approach is that in addition to supporting regression tasks like cost estimation, the zero-shot models can support a broad range of different optimization tasks like MV selection, index selection, or query optimization, etc. and similarly generalize for unseen databases out-of-the-box, simultaneously. The core idea is that the optimization procedure navigates the search space and uses a zero-shot cost model to select an instance in the search space. For example, for MV selection the zero-shot cost model would be used to evaluate the runtime of a set of MV candidates that are enumerated by an optimization procedure. Similarly, one could also employ zero-shot cost models for index selection or query optimization.

Overall, the two core components which are required for this direction are the selection of an appropriate optimization procedure as well as the extension of the zero-shot model to evaluate the cost for an instance in the search space, which we call zero-shot models that provide a *what-if mode*. To demonstrate the ability of this approach, let us take a concrete example of learned MV selection where we augment the zero-shot cost model by a what-if mode that estimates query runtime for a given set of MV candidates. Moreover, we combine the extended zero-shot cost model with an optimizer component (cf. Figure 6). We first elaborate now on the optimizer before we then explain how the zero-shot model is extended by a what-if mode.

The optimizer component of our approach is composed of three main sub-components: *(i)* a *candidate generator*, *(ii)* a *what-if query rewriter*, and *(iii)* a *search procedure*. The candidate generator is the component that gets as input the database and workload for which materialized views should be generated along with certain restrictions (e.g., constraints on the maximum number of generated views). Based on this information, the candidate generator enumerates different so-called MV candidates which are supposed to speed up the workload (i.e., a set of queries) on the given database. As the procedure to enumerate MVs, any existing view enumeration strategy such as [1] or even exhaustive enumeration can be used. For each MV candidate, the what-if rewriter is called which produces query plans of the workload queries where the MV candidates are applied (e.g., sub-plans are replaced by MVs as shown in Figure 3 (b)). Once the plans are rewritten, the query plans, which now include the MV candidates are handed over to a zero-shot model to estimate the cost of the query for the given MV candidate. That way, the candidate generator together with the what-if rewriter enumerate a large set of MV candidates and for each of those, the cost for every query of the workload for the given MV is estimated. The search procedure finally picks the different MV candidates forming the set of MVs which results in the best overall runtime of the workload on the given database.

Key to this optimization procedure is that the zero-shot cost model is able to predict the cost for a query with a given materialized view candidate. For this, as mentioned before, the query rewriter produces query plans of the workload where the MV candidates are applied (e.g., sub-plans are replaced by MVs as shown in Figure 3(b)). To enable the query optimizer to predict cost with plans that include MVs instead of subplans, the zero-shot model needs to be extended to encode MVs as part of its graph representation. To be more precise, for this we extended the graph representation by a new node type for MVs as shown in Figure 3(b), which can take transferable features of MVs such as row width but also the number of rows as input. That way, zero-shot models can predict the cost of queries that involve a MV instead of a sub-plan with multiple tables. Moreover, the zero-shot cost model was trained with additional training data such that the zero-shot cost model can predict query cost not only for queries without MVs but also for queries including MVs.

5.2 Case Study and Results

To show the benefit of our approach we conduct a case study where we perform the MV selection task as described before using an extended zero-shot cost model that can predict query runtime for the plans which include MVs. In the following, we present the performance of our approach on various unseen databases and workloads (cf. Figure 7).

Setup and Benchmarks. In this case study, we use six real-world data sets and a generated set of queries that represent typical OLAP queries (e.g., select-join-aggregate queries) as workload. The details of the data sets and the workload can be found in [10]. For comparison, we use an approach called cost-based merging that is implemented in MS SQL server [1] as baseline. Cost-based merging is based on simple heuristics for view enumeration and the cost estimates from the SQL Server to select the set of views. Moreover, to show how well the individual approaches work we show the runtime also for the optimal set of MVs. As a metric, to show the efficiency of our approach compared to cost-based merging, we used relative runtime which is defined as:

$$\frac{\textit{query runtime with MV}}{\textit{query runtime without MV}}$$

Finally, for our approach (Optimization w. Zero-shot Costs) it is important to note that the zero-shot model is trained once and used out-of-the-box for each of the six different data sets (i.e., without seeing training queries and MVs for the database for which we aim to select the optimal set of MVs).

Initial Results. Our experiments show that in general, the baseline (cost-based merging) is unable to gain major speedups on most of the datasets. Only on the *walmart* benchmark, a reduction in the runtime of the workload of more than 50% is possible. In contrast to the baseline, though, our zero-shot optimization approach chooses a set of MVs that enable a much lower relative runtime in comparison to the baseline [1] for all the data sets as we can see (cf. Figure 7). An important observation is that with the increasing view limit, we see a substantial improvement in the relative runtime and the performance converge to the optimum of our approach. The reason here is that with a limited set of MVs, selecting a MV which is sub-optimal will lead to an inferior runtime of the complete workload. However, if more MVs can be selected, the effect of such a sub-optimal selection does not have much of a high impact. Furthermore, on some data sets (e.g., JOB-light and walmart) the performance of our approach even is very close to the optimal selection of MVs for all view limits showing that zero-shot cost models can precisely predict the expected benefits of a workload given a set of MVs.

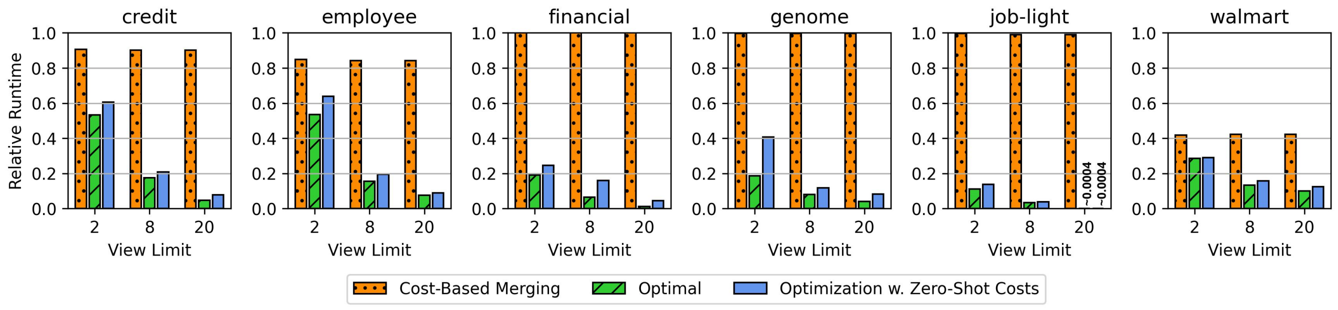


Figure 7: Relative runtime improvement of different MV selection strategies over running the workload without MVs. As we can see, our approach exceeds the performance of the baseline [1] on all the data sets. Another important observation is that the performance further improves with the increase in view limit and thus converging to the optimum.

6 TOWARDS END-TO-END LEARNING

While the aforementioned approaches have several advantages such as they can be used with only minor modifications to the zero-shot cost models for solving additional tasks, they are still restricted in several aspects. To be more precise, the first direction of fine tuning can only solve related tasks to the zero-shot cost models, e.g., estimating additional cost metrics, and the second direction of combining with optimization faces challenges if search spaces become too large, e.g., MV selection for a database with many tables and complex queries. In this situation often simplifying heuristics are used to prune the search space, which can lead to non-optimal results. For example, [1] uses a simple rule-based enumeration strategy instead of exhaustive enumeration.

Therefore, we propose a third direction of end-to-end zero-shot models. Here, the main difference in contrast to navigating huge search spaces with heuristics and finding an optimum based on a zero-shot cost model, is that the end-to-end zero-shot model will itself learn to navigate the search space and thus can come up directly with a solution without the need for involving additional optimization procedures. In the following, as this direction is a part of our ongoing research, we cannot provide any initial results yet in this paper but instead, we discuss the promises and challenges of this direction that we are currently facing to enable end-to-end zero shot models.

Promises. This approach does therefore not need any simplifying assumptions to quickly come up with good solutions or prune the search space but instead learns to enumerate candidates as part of the model. Again, important is that the zero-shot end-to-end model can be used out-of-the-box for an unseen database. Moreover, in addition to being able to navigate large search spaces, we believe that training an end-to-end zero shot model directly for the task at hand, instead of calculating a cost value that is used to select an ideal candidate along with a search procedure, provides several other benefits. For instance, in the case of selecting MVs by using a cost estimator followed by a search procedure, even with a good cost model the results might achieve only limited performance because good overall cost estimates do not necessarily trade-off alternative MV candidates effectively and enable its usage as adequate comparison functions. Thus, training a model directly for the target task omits these challenges and has the potential to achieve even better results.

Open Challenges. While we believe that an end-to-end zero-shot model is a very promising direction, this direction comes with some major challenges. First, we have to collect additional training data for the previously unseen tasks e.g., runtimes of different physical execution plans in case of query optimization. Second, finding a representation of the design space which captures sufficient information to enable a zero-shot model to navigate a search space is often non-trivial. For instance, for query optimization, the join order is a hard sub-problem since it depends on the size of intermediate joins which are hard to predict a priori. In particular, the optimal join order depends on the data distribution of the underlying database, where in case two columns are correlated, the result sizes might be larger. Hence, this has to be encoded in the search space representation. While zero-shot cost models alleviate this problem by taking cardinality estimates as input to the model, this is not always a viable solution, since for instance for join ordering, we cannot predict cardinalities for all possible join orderings. In addition, it is challenging to find the training objective for the model we are interested in. For instance, for cost estimation, the model can be trained directly to minimize the deviation of the prediction from the actual latency whereas, in this approach, the training objective should model how efficient the point in the design space is. A natural fit for this problem formulation could be reinforcement learning which has previously also been used for workload-driven learned query optimizers.

Discussion. Overall, this shows, that there is no silver bullet in coming up with an end-to-end zero-shot model for high-level database tasks and instead requires designing an end-to-end zero-shot model individually for each task. Despite the challenges of this approach, we believe that it has significant potential since it could enable zero-shot models which are tailored to a specific task.

7 RELATED WORK

Recently many learned DBMS components have been proposed. In the following, we discuss related work starting with learned cost models. Afterwards, we discuss how learning has been used for other components beyond cost models. Finally, we discuss recent other trends on multi-task learning and transfer learning that are highly related to this work presented in this paper.

Learned Cost Models. Cost estimators, like our previously mentioned zero-shot approach, are hereby only one of the many fields where learned models play an increasingly important role. For cost estimation, plan-structured neural prediction models [27, 30] have been proposed featurizing the physical query plan as a tree. However, the workload-driven paradigm is predominantly used in these approaches and thus require retraining with thousands of query execution for unseen databases. In contrast, zero-shot learning extenuates the need to run a representative workload for new databases completely by design. Further approaches extend workload-driven models by improving inference and training performance [14] and enable concurrent query latency predictions [36]. Earlier work on predicting the cost of queries proposed the use of statistical methods to learn models at per-operator level [2, 20]. However, these models are too simplistic and do not learn the interactions of operators and therefore achieve performance inferior to workload-driven approaches.

Other Learned Components. Similarly, in the field of query optimizers, many approaches, most of them based on Reinforcement Learning (RL), follow the workload-driven paradigm including query optimizers using value iteration like Neo [26], Bao [24] or Balsa [33] as well as other RL based query optimizers like Krishnan et al. [18] and Marcus et al. [25]. The field of cardinality estimation draws a different picture. Several data-driven approaches like DeepDB [13] and Neurocard [34] have been proposed, which extract data characteristics directly from the dataset without the need to execute queries. To model the data characteristics, DeepDB is using Sum-Product Networks while Neurocard is built on deep autoregressive models. Both show their superiority over supervised query-driven estimators like MSCN [16]. Besides mentioned areas, there has also been a lot of work in the fields of MV selection [7, 21], index selection [4, 19], query scheduling [23, 29], knob-tuning [35] and partitioning [12].

Recent Directions. Recently, a few new approaches for multi-task and transfer learning have been proposed. For example, [32] has proposed a vision of a transferable model architecture, which can be used for solving many different database tasks such as instance cardinality estimation, cost estimation, query optimization, or scheduling as well as different databases. To achieve the transfer to different tasks and databases, i.e., to enable that the model is task- and database-independent, they propose a database-agnostic and task-independent shared representation model, which captures the underlying data, query and further inputs and passes the representation to subsequent task-specific models. While this direction is highly related to our approach, the initial vision paper only targets tasks related to query optimization (e.g., cardinality and cost estimation, or join order selection) while our paper aims to provide a transferable solution that can be used for a much broader set of tasks. Another approach, that follows a similar idea of creating a model with a transferable for datasets in order to achieve database independence, is proposed in [22]. The main contribution of this paper is a pre-trained summarization model, which can be applied for other datasets similar to pre-trained models for text like BERT [3]. These summarization models can then be used as input for subsequent models like cardinality estimators to work out-of-the-box on unseen databases. Again similar to [32], while transferable across databases, [22] is limited for which tasks it can be used and in fact the paper only shows the applicability of the model for the task of cardinality estimation across databases.

8 CONCLUSION

In this paper, we have shown that extending the zero-shot cost estimation model for further tasks is possible in multiple ways. We proposed three general directions to extend zero-shot models to both — related regression tasks as well as more high-level database tasks. To show the potential of described directions, we applied the individual concepts in different case studies and showed initial promising results. Most importantly, with the case studies and the initial results we could show that zero-shot models when generalized to other tasks can still achieve accurate and robust results on unseen databases and provide a performance similar to or even better than state-of-the-art workload-driven approaches.

Clearly, while the initial results are promising, there are still many open questions. First, we aim to show that zero-shot models using these directions can support a really wide range of different tasks beyond the case studies shown in this paper. Moreover, while the direction of end-to-end learning zero-shot models seems interesting, we still need to show that this direction can lead to models that can solve complex tasks on unseen databases out-of-the-box.

ACKNOWLEDGMENTS

We thank the reviewers for their feedback. This research is funded by the BMBF project within the “The Future of Value Creation – Research on Production, Services and Work” program, the Hochtief project *AICO* (AI in Construction), the HMWK cluster project *3AI* (The Third Wave of AI), as well as the DFG Collaborative Research Center 1053 (MAKI). Finally, we want to thank hessian.AI at TU Darmstadt as well as DFKI Darmstadt.

REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. 2000. Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB*, Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang (Eds.). Morgan Kaufmann, 496–505.
[2] Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. 2012. Learning-based Query Performance Modeling and Prediction. In *ICDE*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, 390–401.
[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018).
[4] 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*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 969–984.
[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. 1126–1135.
[6] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *ICDE*. 592–603.
[7] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. 2021. An Autonomous Materialized View Management System with Deep Reinforcement Learning. In *ICDE*. IEEE, 2159–2164.
[8] Roman Heinrich, Manisha Luthra, Harald Kornmayer, and Carsten Binnig. 2022. Zero-shot cost models for distributed stream processing. In *DEBS*. ACM, 85–90. https://doi.org/10.1145/3524860.3539639
[9] Benjamin Hilprecht and Carsten Binnig. 2022. One Model to Rule them All: Towards Zero-Shot Learning for Databases. In *CIDR*. www.cidrdb.org.
[10] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *Proc. VLDB Endow.* 15, 11 (2022), 2361–2374. https://doi.org/10.14778/3551793.3551799
[11] Benjamin Hilprecht, Carsten Binnig, Tiemo Bang, Muhammad El-Hindi, Benjamin Hättasch, Aditya Khanna, Robin Rehrmann, Uwe Röhm, Andreas Schmidt,

Lasse Thostrup, and Tobias Ziegler. 2020. DBMS Fitting: Why should we learn what we already know?. In *CIDR*. www.cidrdb.org.

[12] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. 2020. Learning a Partitioning Advisor for Cloud Databases. In *SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 143–157.

[13] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *PVLDB* 13, 7 (2020), 992–1005.

[14] Johan Kok Zhi Kang, Gaurav, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. 2021. Efficient Deep Learning Pipelines for Accurate Cost Estimations Over Large Scale Query Workload. In *SIGMOD*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1014–1022.

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. [n.d.]. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NeurIPS*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.).

[16] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[17] Jan Kossmann, Alexander Kastius, and Rainer Schlosser. 2022. SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning. In *EDBT*, Julia Stoyanovich, Jens Teubner, Paolo Guagliardo, Milos Nikolic, Andreas Pieris, Jan Mühlig, Fatma Özcan, Sebastian Schelter, H. V. Jagadish, and Meihui Zhang (Eds.). OpenProceedings.org, 2:155–2:168.

[18] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018).

[19] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2020. An Index Advisor Using Deep Reinforcement Learning. In *CIKM*, Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux (Eds.). ACM, 2105–2108.

[20] Jiexing Li, Arnd Christian König, Vivek R. Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. *CoRR* abs/1208.0278 (2012).

[21] Xi Liang, Aaron J. Elmore, and Sanjay Krishnan. 2019. Opportunistic View Materialization with Deep Reinforcement Learning. *CoRR* abs/1903.01363 (2019).

[22] Yao Lu, Srikanth Kandula, Arnd Christian König, and Surajit Chaudhuri. 2021. Pre-training Summarization Models of Structured Datasets for Cardinality Estimation. *PVLDB* 15, 3 (2021), 414–426.

[23] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *SIGCOMM*, Jianping Wu and Wendy Hall (Eds.). ACM, 270–288.

[24] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2020. Bao: Learning to Steer Query Optimizers. *CoRR* abs/2004.03814 (2020).

[25] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *aiDM@SIGMOD*, Rajesh Bordawekar and Oded Shmueli (Eds.). ACM, 3:1–3:4.

[26] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *PVLDB* 12, 11 (2019), 1705–1718.

[27] Ryan C. Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *PVLDB* 12, 11 (2019), 1733–1746.

[28] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB* 2, 1 (2009), 982–993.

[29] Yangjun Sheng, Anthony Tomasic, Tieying Zhang, and Andrew Pavlo. 2019. Scheduling OLTP transactions via learned abort prediction. In *aiDM@SIGMOD*. 1:1–1:8.

[30] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *PVLDB* 13, 3 (2019), 307–319. https://doi.org/10.14778/3368289.3368296

[31] Francesco Ventura, Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2021. Expand your Training Limits! Generating Training Data for ML-based Data Management. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. 1865–1878. https://doi.org/10.1145/3448016.3457286

[32] 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. *CoRR* abs/2105.02418 (2021).

[33] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. *CoRR* abs/2201.01441 (2022).

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

[35] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 415–432.

[36] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query Performance Prediction for Concurrent Queries using Graph Embedding. *PVLDB* 13, 9 (2020), 1416–1428.