

One Model to Rule them All: Towards Zero-Shot Learning for Databases

Benjamin Hilprecht
TU Darmstadt

Carsten Binnig
TU Darmstadt

ABSTRACT

In this paper, we present our vision of so called *zero-shot learning for databases* which is a new learning approach for database components. Zero-shot learning for databases is inspired by recent advances in transfer learning of models such as GPT-3 and can support a new database out-of-the box without the need to train a new model. As a first concrete contribution in this paper, we show the feasibility of **zero-shot learning for the task of physical cost estimation and present very promising initial results**. Moreover, as a second contribution we discuss the core challenges related to zero-shot learning for databases and present a roadmap to extend zero-shot learning towards many other tasks beyond cost estimation or even beyond classical database systems and workloads.

1 INTRODUCTION

Motivation. Building computer systems often involves solving complex high-dimensional combinatorial problems in all layers of those systems. To reduce complexity when building those computer systems and solve the problems in a tractable manner, these systems have heavily relied on heuristics or simplified analytical models in the past. Very recent work in the systems community, however, has outlined a broad scope where machine learning vastly outperforms these traditional heuristics. This is also the case for databases, where existing DBMS components have been replaced with learned counterparts such as learned cost and cardinality estimation models [9, 11, 13, 23, 28, 31] as well as learned query optimizers [15, 20–22] or even learned indexes [6–8, 14] and learned query scheduling strategies [19, 27].

The predominant approach that has been used in the past for learned database components is workload-driven learning. The idea of workload-driven learning is to **capture the behavior of a DBMS component by running a representative set of queries over a given database and then use the observations to train the underlying model**. For example, for learned cardinality estimation models such as [13, 28] a set of queries must be executed to collect query plans and their cardinalities, which serve as training data for learning a **model that can be used to estimate the cardinalities for new queries**. The very same procedure is applied if workload-driven learning is used for the other DBMS components such as learned physical design advisors (e.g., an advisor for index selection) [10, 16, 18, 34] or other components.

However, a major obstacle to these workload-driven approaches is the collection of training data. For example, in [13, 28] it was shown that thousands of query plans and their true cardinalities are needed for training the model to achieve a high accuracy. Running such a set of training queries on potentially very large databases to collect the training data can take hours or even days while the actual training of the underlying models often only takes a few

minutes. And unfortunately, the training data collection needs to be repeated for every new database that needs to be supported.

To reduce the high cost of training data collection, reinforcement learning (RL) has been used to execute training queries [10, 17, 18, 34] in a more targeted manner (i.e., letting the RL agent decide which queries to execute next). However, even with reinforcement learning still a large amount of training queries needs to be executed for learning a model. Moreover, training the model is not a one-time effort since similar to workload-driven approaches the learning procedure needs to be repeated for every new database at hand.

A different direction that has thus been proposed to avoid the expensive training data collection by running queries on a new database are so called data-driven approaches [11, 31, 32] that learn a model purely from the underlying data. A prime example where data-driven learning is a perfect fit is cardinality estimation. However, data-driven learning is no silver bullet either since for some DBMS components the information about the runtime behavior of queries is required. One such example is learned physical cost estimation where the runtime behavior of queries needs to be captured by a model to make predictions. A similar observation holds for many other database tasks such as physical design tuning or knob tuning where the effects of a certain decision on the runtime of a workload need to be learned.

Vision and Contributions. In this paper, we thus present our vision of so called *zero-shot learning for databases* which is a new learning approach for database components that can support a broad set of tasks on the one hand but does not require to collect training data for supporting a new database on the other hand. In that regard, zero-shot learning for databases combines the benefits of data-driven learning and workload-driven learning. The general idea behind zero-shot learning for databases is motivated by recent advances in transfer learning of models. Similar to other approaches such as GPT-3 [3] which enables zero-shot learning for NLP, a zero-shot model for databases is trained on a wide collection of different databases and workloads and can thus generalize to a completely new database and workload without the need to be trained particularly on that database.

As a core contribution in this paper, we discuss how such an approach of zero-shot learning for databases could work and we also show the feasibility of this approach for the task of physical cost estimation. In our initial results for physical cost estimation, we show that zero-shot models can significantly outperform workload-driven approaches even when providing workload-driven models with a large number of training queries for a particular database at hand whereas zero-shot models can support them out-of-the-box. Moreover, we believe that the real power of zero-shot learning stems from the fact that it is a general principle that can be used for various learned database tasks. For example, we already have initial promising results that show zero-shot learning can not only

be used for physical cost estimation on a new database but also for physical design tuning and, in particular, index selection on a database the model has not seen before.

Outline. The remainder of this paper is structured as follows: Section 2 first gives an overview of zero-shot learning for databases and discusses the core challenges related to zero-shot learning. Section 3 then discusses the case study of using our approach for learning a zero-shot physical cost model. Moreover, in this section we also present our initial experimental results. Afterwards, Section 4 discusses a research roadmap for zero-shot learning for databases beyond cost estimation. Finally, we conclude with a peak into the future in Section 5.

2 ZERO-SHOT LEARNING FOR DATABASES

In this section, we first give a brief overview of how zero-shot learning for databases works in general and then discuss the core challenges related to enable this approach in an efficient manner.

2.1 Overview of the Approach

Figure 1 shows the high-level idea that is behind our vision of zero-shot learning for databases. During the learning phase, similar to workload-driven learning, for zero-shot learning we have to execute a representative workload and collect training data.

The main difference to workload-driven learning though, which makes zero-shot learning attractive, is that a zero-shot model is database-independent. This allows a zero-shot model to generalize to unseen databases out-of-the-box. To allow a zero-shot model to generalize to new databases without the need to retrain the model for this particular database, we provide a new method for featurizing queries as we discuss below (cf. Key Challenges). This new featurization method is at the core of learning zero-shot models in a database-independent manner and thus enables them to make predictions for queries on a new database (e.g., for physical cost estimation) that the model has never seen before.

Moreover, for being able to generalize to new databases, a zero-shot model is trained on different databases. While this might seem to cause high upfront costs before a zero-shot model can be used, it is important to note that the *training data collection is a one-time-effort* which is very different from workload-driven learning that needs to collect training data for every new database a model should support. Moreover, cloud database providers such as AWS, Microsoft, or Google, typically already have significant amounts of such information available since they keep logs of their customer workloads and could thus apply zero-shot learning right away without the need to collect training data in the first place.

Finally, a last important aspect is that zero-shot learning is not only database-independent but is a general learning approach that can be applied to a *variety of database tasks* that range from physical cost estimation, design tuning or knob tuning to query optimization and scheduling as we discuss in Section 3 and Section 4. To enable zero-shot models to generalize to different tasks though, the models need to be capable of capturing not only information about query plans and their runtimes but also information about other aspects (e.g., how indexes or changes in the database configuration influence the query runtime) as we discuss later.

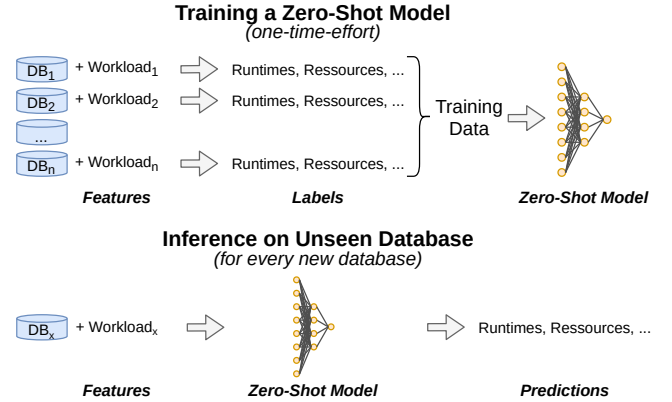


Figure 1: Overview of zero-shot learning for databases. In line with other zero-shot approaches such as GPT-3 which enables zero-shot learning for NLP, a zero-shot model for databases can generalize to a completely new database and workload without the need to be trained on that particular database.

2.2 Key Challenges

Enabling zero-shot learning for databases comes with various research challenges. In the following, we discuss the key challenges that we think are at the core to make zero-shot learning for databases efficient and accurate.

Featurization of Database and Queries. A core challenge of zero-shot learning is how to featurize a query of a workload in a flexible (i.e., database-independent) manner to capture the general runtime behavior of queries. While workload-driven approaches also need to featurize queries, they typically do this in a static and database-dependent manner. Early approaches such as [13], for example, use a simple (static) one-hot encoding as shown in Figure 2 (left-hand side). This featurization, however, assumes that one particular database schema is given and all possible query patterns (join paths, selection predicates, etc.) are known in advance. Consequently, such a scheme only allows to encode a fixed set of queries over a fixed database.

For zero-shot learning instead, we use a much more flexible approach to be able to express general database queries over arbitrary databases. The main idea is that we featurize the query plan using a tree-based encoding, where nodes represent plan operators as shown in Figure 2 (right-hand side). Tree-based encodings have already been used for more recent workload-driven approaches such as [28]. However, the tree-based encodings for workload-driven approaches still use a database-dependent one-hot encoding to express the features of a tree node (e.g., the involved tables). Consequently, only query plans over a fixed database can be expressed.

Different from such a database-dependent tree encoding, our zero-shot encoding uses a tree-based encoding where features of nodes are database-independent (i.e., they can be derived from any database at hand). As a result, queries over different databases can be expressed as input to a zero-shot model. For example, typical database-independent features that can be derived from any database are features such as the tuple width for operators (input and output) or information about the table scanning overhead of the tables involved in a query plan (e.g., as number of database pages).

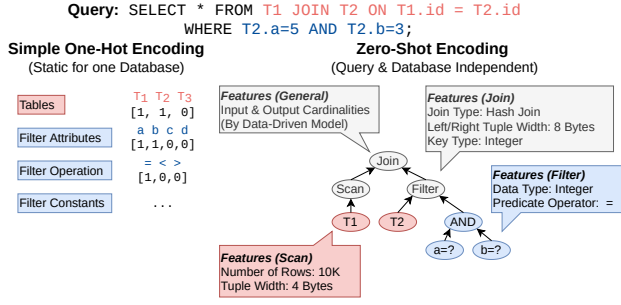


Figure 2: Overview of different featurization methods. Simple one-hot encoding can only support a fixed set of queries over a given database schema. Zero-shot learning uses a more flexible featurization that allows to encode arbitrary query plans over unseen databases using a tree-based encoding with nodes representing database operators that take database-independent features as input.

A first concrete implementation of such a flexible encoding will be presented in the next Section 3 for our case study.

Finally, in Section 4, we discuss also first ideas of how our zero-shot featurization can be extended to also include information about other aspects such the physical design (e.g., the indexes used). However, a key question is how to represent these additional features in an efficient manner. For example, database configurations of commercial systems can have hundreds to thousands of parameters. One interesting direction to tackle this problem could be to compute *database embeddings* that are able to express the similarity of two database configurations by the distance in the embedding space and thus allow zero-shot models to generalize in a robust manner even for a broader set of tasks.

Training Data and Uncertainty. A second key challenge to enable zero-shot learning is clearly the training data collection for learning a zero-shot cost model. Here an important question is how many and which databases and workloads a zero-shot model needs to observe during training to make robust decisions on unseen databases. As we show in our initial experiments for zero-shot cost estimation in Section 3, for example, already a relatively small number of databases along with the respective workload information (e.g., the featurized query plans and runtimes) is sufficient to generalize robustly and even outperform existing workload-driven approaches. However, clearly we need to develop more theoretical foundations to help guide us as to whether or not a zero-shot model has seen sufficiently many databases and queries.

This question is closely related to estimating how uncertain a zero-shot model is in its prediction (e.g., by computing **confidence intervals** for its **predictions**). Such uncertainty estimates could then be used in various ways. One way to use them is to decide during training when a zero-shot model has seen enough training data and for which queries and databases a model is still uncertain. This information can then be also used to create further training data in a more targeted manner that make the model more certain for those cases. Another direction could also be to use these uncertainty estimates at runtime (e.g., to decide when a model is likely to be wrong) and eventually fall back to traditional heuristics.

A related question in this respect is how to create the training databases and workloads if they are not yet available (as for

cloud providers). An interesting direction here is to use a synthetic approach and generate databases / workloads with different characteristics.

Separation of Concerns. Finally, a last important aspect of zero-shot models is to decide what should be learned by the models to fulfill its core promise and when to separate concerns. For example, workload-driven approaches often prefer end-to-end learning which captures both data- and system-characteristics in a single model. However, this results in models that do not generalize to new databases with different data characteristics.

To put it differently, in order to make predictions for a query plan (e.g., runtime), workload-driven models typically internalize both the data characteristics (e.g., data size and distributions) as well as the system characteristics (e.g., the runtime complexity of database operators) in one model. However, since the data characteristics can be entirely different for a new database, such an end-to-end approach will not work for zero-shot learning. Hence, we suggest that data characteristics for zero-shot learning should be captured by separate data-driven models (such as [11, 31]). For example, a database-dependent feature that can be captured by a data-driven model are the input- and output-cardinalities of operators in a query plan. That way, when using cardinalities as input features for the zero-shot models, these models learn to predict the runtime behavior of operators based on input/output sizes that can be derived for any database which enables database-independent predictions of the zero-shot models.

One could now argue that this violates the core promise of zero-shot learning since data-driven models need to be learned for each new database. However, data-driven models can be derived from a database without running any training query and typically a sample of the database is enough to train these models. Moreover, for cardinality estimation we could even use simple non-learned estimators (e.g., histograms) as input for the zero-shot models. As we show in our initial results in Section 3, even those simple estimators often provide sufficient evidence for our zero-shot cost models to produce accurate estimates.

To summarize, a key question in this context is to decide what a zero-shot model should learn and which aspects should be treated separately. Clearly a guide for this question is to think about what is database-dependent and what is database-independent and only include the database-independent aspects in a zero-shot model. Moreover, as we discuss later, for design tuning or query optimization another question is how to combine zero-shot models with optimization procedures or other learning approaches (e.g., value networks) to implement efficient search strategies.

3 CASE STUDY: COST ESTIMATION

In this case study, we demonstrate how zero-shot learning can be used for physical cost estimation. We envision this to be a potential core building block for zero-shot models for many other DBMS tasks as we discuss in the next section.

3.1 Zero-Shot Cost Estimation

The main promise of zero-shot cost estimation is that a trained model can predict the query runtime on a new database out-of-the-box. In the following, we give a brief overview of how we implemented our initial prototype for zero-shot cost estimation for

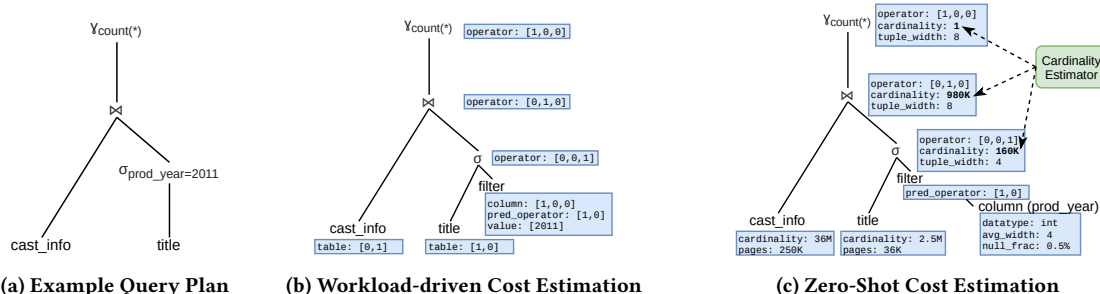


Figure 3: Learned Cost Estimation. (a) Example query plan. (b) Workload-driven cost model which features the plan. However, tables and columns are one-hot encoded and can thus only be used for the current database. (c) In contrast, in a zero-shot model the features are database-independent and thus the resulting model can be reused for different databases.

Postgres and contrast it to recent workload-driven approaches for cost estimation using the example query plan in Figure 3a.

As shown in Figure 3c, we encode query features as a tree. Each node in this tree represents a physical operator as opposed to a logical operator to capture the differences in runtime complexity during learning. Moreover, as mentioned before, each node uses database-independent features. For example, the involved tables (title and cast_info) and columns (prod_year) are featurized via database-independent characteristics (i.e., that can be derived from any database at hand). For example, for a table scan we use features such as the number of pages of a table or information about the data type and average width in bytes for each column that allows the model to learn the runtime complexity of the scan independent of the concrete database at hand. Note, this encoding is different from recent workload-driven approaches that also use a tree-based encoding as shown in Figure 3b since these use a one-hot encoding which is database-dependent. For example, the table nodes only encode which tables are used by simply referring to the according table name in the schema.

Moreover, as mentioned before, for zero-shot models we want the model to learn the general runtime behavior of operators in a DBMS (i.e., system characteristics) in a database-independent manner. Hence, the zero-shot model should learn system characteristics separate from data characteristics. This is very different from workload-driven models which learn both aspects end-to-end in one model as mentioned before. This can also be seen by the fact that workload-driven models include the values involved in filter predicates (e.g., 2011) in the featurization of a query and thus learn the selectivity of the filter operation implicitly. Different from this, for zero-shot models we only encode the predicate structure (to represent the general computational complexity) and explicitly take the cardinality from a data-driven model as input.

Finally, a last important aspect is our proposed model architecture as well as the learning and inference procedure for a featurized plan as shown in Figure 3c. The learning overall happens in three steps: we first encode the features of the individual nodes in a fixed-size hidden vector (the initial hidden states). The hidden states of the individual plan nodes are afterwards combined using a message passing phase in the tree. Finally, the hidden state of the top node is fed into a multilayer perceptron (MLP) which predicts the final runtime. In particular in the message passing phase, the tree is traversed bottom up. The hidden states of the children are summed up (similar to the DeepSets [33] architecture) and combined with

the hidden state of the parent node using an MLP to update the hidden state. This process is repeated until the top node is reached and the information of the entire tree is combined. For inference to make a prediction for a query plan on a new database, a new tree is constructed using the node-specific MLPs and the features are propagated through the tree up to the root node which then predicts the runtime for the new query.

3.2 Initial Evaluation

Setup and Baselines. In an initial experiment, we trained a zero-shot cost estimation model on workloads we executed on a set of 19 publicly available datasets that cover a range of databases with a different number of tables and database sizes. We then predicted the runtimes for a workload on an entirely unseen database to validate that zero-shot cost estimation can provide high accuracies.

As baselines, we used state-of-the-art workload-driven approaches for cost estimation. In particular, we used the end-to-end model of [28] called E2E and the MSCN architecture [13] that was initially proposed for cardinality estimation. In addition, we use a simple linear model that obtains actual runtimes from the internal cost metric of the Postgres optimizer (called Scaled Optimizer Cost).

Training of Baselines and Zero-Shot Models. For training the workload-driven models, we collected training data of different sizes ranging from a small training set size of 100 queries up to very large training sets with 50,000 queries (similar to [28]). Note, that for every new database such training queries need to be executed and the runtimes need to be collected before a workload-driven model can be trained. Moreover, if the database is updated and data characteristics change, the training data collection needs to be repeated.

For training the zero-shot model, we also need to collect training data. Overall, this gathering of the training data is clearly a significant effort. However, as mentioned before this is a one-time effort and the resulting model can be reused across different databases. To be more precise, for zero-shot learning we randomly generated 5,000 training queries for each of the 19 databases. In total the workload size for zero-shot learning was thus in a similar ballpark compared to the maximum size we used for training the workload-driven baselines. Moreover, the queries for zero-shot learning and workload-driven learning were similar and covered up to five-way joins with up to five numerical and categorical predicates and up to three aggregates. However, different from a workload-driven model,

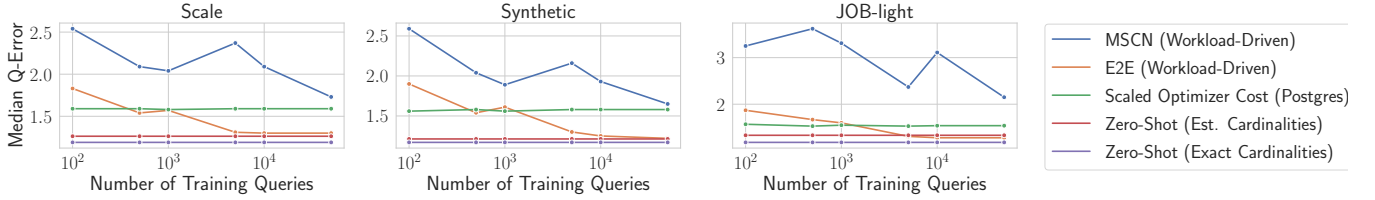


Figure 4: Accuracy of Cost Estimation of Zero-Shot and Workload-Driven Models for IMDB Benchmarks. The zero-shot models were trained using query executions on entirely different databases and thus do not require any queries on the IMDB database. In contrast, the workload-driven models require executions on every database it needs to support - i.e., the IMDB database for this experiment. For the workload-driven models we vary the number of training queries as shown on the logarithmic x-axis. Overall, workload-driven models are less accurate than zero-shot models even after a significant number of queries.

the zero-shot model was not trained on the database it should make a prediction on.

Initial Results. To evaluate the trained models, we used our zero-shot model as well as the other baselines to predict the runtimes of the commonly used *scale*, *synthetic* and *job-light* benchmarks [13] on the IMDB database. For zero-shot models, we show two versions: one that uses the Postgres cardinalities as input as well as one version, which uses exact cardinalities (as an upper baseline to show how accurate zero-shot models can become).

As a result, we report the commonly used Q-error which is the factor the predicted runtime deviates from the true runtime. The advantage of zero-shot learning over workload-driven approaches is that no queries on the test database are required for training. To demonstrate this tradeoff, we vary the number of training queries that can be used for the workload-driven baselines and compare the accuracy with zero-shot learning in Figure 4. Overall, zero-shot learning can predict the runtimes very accurately. Since the IMDB dataset was never used for one of the training queries this shows, that zero-shot learning can generalize to unseen databases. Interestingly, even the zero-shot model using only cardinality estimates of the Postgres optimizer is still very accurate.

In contrast, the workload-driven E2E models [28] are less accurate than zero-shot models even for a large set of training queries on the IMDB database for the *scale* and *synthetic* benchmarks. For the simpler *job-light* queries that rarely contain range predicates, the E2E model is on par with the zero-shot model for the larger training sets. However, still the E2E models cannot match the performance of zero-shot learning with exact cardinalities. In addition, we note that the MSCN models are significantly less accurate since they use a much simpler featurization based on one-hot encodings (and not a tree-based featurization). Moreover, note that even though we repeated all measurements multiple times and report the median there are still some peaks in the reported Q-errors of MSCN due to a particularly high variance.

While our initial results are promising, in future we plan to conduct more extensive experiments that show the robustness of zero-shot learning in several dimensions (e.g., more complex queries but also other databases).

4 BEYOND COST ESTIMATION

In the following, we will discuss how zero-shot models can be extended beyond cost estimation.

4.1 Physical Design and Knob Tuning

A first clear extension of zero-shot cost models as described in Section 3 is to allow them to support a so called “*What-If*” mode. This enables zero-cost models to predict the runtime of a query given a certain physical design or a database configuration (also called knob configuration). For example, one could then ask the model how the runtime of a query changes if a certain index would exist or how the runtime changes if the buffer size would be increased.

Both these tasks — physical design and knob tuning — are classical problems of DBMSs that have been addressed in the past already by using optimization approaches [1, 4, 5, 24, 25]. However, the main problem was that these approaches relied on inexact cost estimates coming from classical optimizer cost models that were extended to support a “*What-If*” mode. For that reason, recent approaches have suggested to use workload-driven learning — in many cases reinforcement learning [2, 10, 16, 18, 26, 30, 34]. While these approaches have been shown to be more accurate than the more classical approaches, they again need to first run training queries under different physical layouts or knob configurations for every new database.

Hence, to avoid these high-training costs for every new database one could use a zero-shot cost model in a “*What-If*” mode. To show the feasibility of this direction, we extended our zero-shot cost model to support also decisions towards index tuning. At the core, the zero-shot model for index tuning should be able to support predictions of the runtime as if a certain index would exist in a database. For training a zero-shot cost model that can answer such questions, we again used the 19 databases as training data that we also used in Section 3. However, we additionally created a random but fixed set of indexes per database before running the training queries. The zero-shot cost model could recognize for which training query an index was used since the physical operators in a query plan change (e.g., an index scan instead of a table scan was used). The zero-shot model could thus learn how the runtime changes for query plans, which use an index scan compared to query plans which do not use an index scan.

For showing that the learned model could estimate the runtime of queries correctly given a certain index, we again use the IMDB database for the evaluation that the zero-shot model had not seen before. For testing, we asked the model to estimate the runtime of queries if an index would exist again for randomly selected attributes of queries. The estimation errors for zero-shot learning for this workload are given in Table 1 (last line). As we can see, the estimations are still very accurate but clearly the maximum

Workload	Zero-Shot (Exact Card.)			Zero-Shot (Estimated Card.)		
	median	95th	max	median	95th	max
Scale	1.19	1.93	3.93	1.26	2.46	4.70
Synthetic	1.17	1.90	4.40	1.21	2.17	6.88
JOB-light	1.18	1.85	2.47	1.33	2.56	4.00
Index	1.21	2.51	10.73	1.33	3.59	24.62

Table 1: Estimation errors (Q-errors) of zero-shot models for index tuning (last line) compared to zero-shot cost models without *What-if* support (upper lines).

Q-error increases compared to the results for zero-shot cost models in Section 3 before (upper lines).

To further improve the quality of zero-shot cost models for index tuning one might need to think about a more sophisticated workload sampling or provide additional characteristics for indexes (e.g., expected index height) as input features to a zero-shot model that could be derived with additional data-driven models. Furthermore, we think that we could use zero-shot models to build other design advisors (e.g., for materialized views) or support zero-shot knob tuning to select an optimal database configuration for a given workload without having seen the database for training. Finally, for knob tuning one could also think about using zero-shot models to only guide the search initially to a good start configuration (i.e., to narrow down the search space) and then use online approaches for fine-tuning the knobs since knobs can be changed easily compared to the high cost of changing a physical layout.

4.2 Query Optimization

Another direction for zero-shot learning are learned optimizers. Recently, it was also proposed to replace query optimizers that typically rely on heuristics (i.e., simple cost models) and manual engineering by machine learning models [15, 20–22]. While initial results are promising and even commercial optimizers can be outperformed by learned ones, current approaches are also dominated by reinforcement learning or workload-driven learning in general. Again, all these approaches are database-dependent and cannot generalize to unseen databases. Moreover, for learning an optimizer a huge number of queries has to be executed in either case to learn what is a good plan for a given query. We envision that this overhead for unseen databases can be eliminated completely using zero-shot learning.

An initial naïve approach for this could be to use the devised zero-shot cost estimation model to evaluate candidate plans and thus better guide the query optimizer to plans with low costs. For instance, zero-shot cost estimation models could be used in conjunction with classical dynamic programming or even approaches like Bao [21]. However, with more sophisticated approaches, we think that zero-shot learning could potentially replace classical heuristics like dynamic programming entirely by devising zero-shot value networks to learn search strategies for query optimization. Value networks [29] have shown to learn policies that involve planning-based reasoning. This way, zero-shot query optimizers could come up with plans that classical optimizers would not have considered while avoiding the burden to run thousands of queries to train the learned optimizer for every new database.

4.3 Discussion

In addition to design advisors, knob tuning, or database optimizers there are many more DBMS components that could benefit from zero-shot learning. For example, zero-shot cost models could be used to predict not only the runtime but also other aspects such as resource consumption and thus be used also for runtime decisions (e.g., query scheduling). Moreover, by extending the features of the “*What-If*” mode, we could also support hardware aspects and predict the runtime of queries on an unseen hardware, e.g., to select an optimal cloud instance for a given workload.

Another interesting question is how zero-shot learning should be integrated into the overall DBMS architecture. Here we envision a route where *zero-shot cost models* as presented in Section 3 form a kind of *central brain* in a DBMS that can be leveraged by various DBMS components that complement such a central component with more targeted models. These additional models could for example be zero-shot models that focus on learning particular search strategies or specific data-driven models to capture interesting data characteristics for a component as we discussed before for index tuning.

5 LOOKING INTO THE FUTURE

In this paper, we have shown a new approach for learned database components that can support new databases without running any training query on that database. While we have focused on single-node databases and classical database workloads in the first place, we believe that zero-shot models can be applied more broadly. One direction are distributed DBMSs where zero-shot models can be extended to support tasks such as to optimize a distributed data layout. Another direction is to extend zero-shot models for other types of data-intensive workloads (e.g., data streaming).

Moreover, when thinking more broadly, zero-shot models seem to also be an attractive model for any system builder and can be also used at various levels of granularity to predict the performance of individual components (e.g., very fine-grained on the data structure and algorithm level) or very coarse-grained (at the system level). For example, when being used for data structures and algorithms, zero-shot models would be an efficient vehicle for self-designing data structures [12]. To conclude, we think that zero-shot learning opens up many avenues of research since it provides not only a more *sustainable* way to build learnable system components but it also seems to be a *general paradigm* that can be applied more broadly and at different levels (as discussed before) for building the next generation of data-intensive systems.

ACKNOWLEDGMENTS

We would like to thank the Amazon Redshift team for the early feedback and the discussions about this topic.

REFERENCES

- [1] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB ’00*, page 496–505, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [2] S. Alabed and E. Yoneki. High-dimensional bayesian optimization with multi-task learning for rocksdb. *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021.

- [3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Nee-lakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [4] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, page 227–238, New York, NY, USA, 2005. Association for Computing Machinery.
- [5] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, page 146–155, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [6] J. Ding, U. F. Minhas, J. Yu, C. Wang, J. Do, Y. Li, H. Zhang, B. Chandramouli, J. Gehrke, D. Kossmann, D. Lomet, and T. Kraska. Alex: An updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 969–984, New York, NY, USA, 2020. Association for Computing Machinery.
- [7] J. Ding, V. Nathan, M. Alizadeh, and T. Kraska. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *Proc. VLDB Endow.*, 14(2):74–86, Oct. 2020.
- [8] A. Galakatos, M. Markovitch, C. Binnig, R. Fonseca, and T. Kraska. Fitting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1189–1206, New York, NY, USA, 2019. Association for Computing Machinery.
- [9] B. Hilprecht, C. Binnig, T. Bang, M. El-Hindi, B. Hättasch, A. Khanna, R. Rehmann, U. Röhm, A. Schmidt, L. Thostup, and T. Ziegler. DBMS fitting: Why should we learn what we already know? In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020.
- [10] B. Hilprecht, C. Binnig, and U. Röhm. Learning a partitioning advisor for cloud databases. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 143–157, New York, NY, USA, 2020. Association for Computing Machinery.
- [11] B. Hilprecht, A. Schmidt, M. Kulesa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *Proc. VLDB Endow.*, 13(7):992–1005, Mar. 2020.
- [12] S. Idreos, K. Zoumpatianos, S. Chatterjee, W. Qin, A. Wasay, B. Hentschel, M. S. Kester, N. Dayan, D. Guo, M. Kang, and Y. Sun. Learning data structure alchemy. *IEEE Data Eng. Bull.*, 42(2):47–58, 2019.
- [13] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*, 2019.
- [14] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504, New York, NY, USA, 2018. ACM.
- [15] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *ArXiv*, abs/1808.03196, 2018.
- [16] H. Lan, Z. Bao, and Y. Peng. An index advisor using deep reinforcement learning. *CIKM '20*, page 2105–2108, New York, NY, USA, 2020. Association for Computing Machinery.
- [17] T. Li, Z. Xu, J. Tang, and Y. Wang. Model-free control for distributed stream data processing using deep reinforcement learning. *Proc. VLDB Endow.*, 11(6):705–718, Feb. 2018.
- [18] X. Liang, A. J. Elmore, and S. Krishnan. Opportunistic view materialization with deep reinforcement learning. *CoRR*, abs/1903.01363, 2019.
- [19] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288, New York, NY, USA, 2019. Association for Computing Machinery.
- [20] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. Bao: Learning to steer query optimizers, 2020.
- [21] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, July 2019.
- [22] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM'18, New York, NY, USA, 2018. Association for Computing Machinery.
- [23] R. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *Proc. VLDB Endow.*, 12(11):1733–1746, July 2019.
- [24] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting dbms. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '05, page 239–248, USA, 2005. IEEE Computer Society.
- [25] R. Nehme and N. Bruno. Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 1137–1148, New York, NY, USA, 2011. Association for Computing Machinery.
- [26] T. Schmied, D. Didona, A. Döring, T. Parnell, and N. Ioannou. Towards a general framework for ml-based self-tuning databases. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, EuroMLSys '21, page 24–30, New York, NY, USA, 2021. Association for Computing Machinery.
- [27] Y. Sheng, A. Tomasic, T. Zhang, and A. Pavlo. Scheduling OLTP transactions via learned abort prediction. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019, pages 1:1–1:8, 2019.
- [28] J. Sun and G. Li. An end-to-end learning-based cost estimator. *CoRR*, abs/1906.02560, 2019.
- [29] A. Tamar, S. Levine, P. Abbeel, Y. Wu, and G. Thomas. Value iteration networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2146–2154, 2016.
- [30] T. Wang, S. Ferlin, and M. Chiesa. Predicting cpu usage for proactive autoscaling. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, EuroMLSys '21, page 31–38, New York, NY, USA, 2021. Association for Computing Machinery.
- [31] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica. Neurocard: One cardinality estimator for all tables. *Proc. VLDB Endow.*, 14(1):61–73, Sept. 2020.
- [32] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. *PVLDB*, 13(3):279–292, 2019.
- [33] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [34] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 415–432, New York, NY, USA, 2019. Association for Computing Machinery.