Towards Foundation Database Models

Johannes Wehrstein
Google

Carsten Binnig
Google

Fatma Özcan
Google

Shobha Vasudevan
Google

Yu Gan
Google

Yawen Wang
Google

Abstract

Recently, machine learning models have been utilized to realize many database tasks in academia and industry. To solve such internal tasks of database systems, the state-of-the-art is one-off models that need to be trained individually per task and even per dataset, which causes extremely high training overheads. In this paper, we argue that a new learning paradigm is needed that moves away from such one-off models towards generalizable models that can be used with only minimal overhead for an unseen dataset on a wide spectrum of tasks. While recently, several advances towards more generalizable models have been made, still no model exists that can generalize across both datasets and tasks. As such, we propose a new direction which we call foundation models for databases which is pre-trained in both task-agnostic and dataset-agnostic manner which makes it possible to use the model with low overhead to solve a wide spectrum of downstream tasks on unseen datasets. In this vision paper, we propose an architecture for such a foundation database model, describe a promising feasibility study with a first prototype of such a model, and discuss the research roadmap to address the open challenges.

1 Introduction

Learning database tasks. Many of the most complex components in database management systems (DBMSs) involve solving non-trivial problems. To tackle them, classical DBMS components rely on heuristics or simplified analytical models, which cause, however, inferior performance of a DBMS in many cases. Hence, in the recent years, the database community has outlined a new direction of so-called learned DBMS components. Examples, where such learned approaches have successfully been used, include a wide spectrum of DBMS components such as learned query optimizers, [7, 9, 12, 14, 19], learned query scheduling strategies, [3, 11, 18], or even learned storage layouts and indexes [2, 5, 8, 17] amongst many other examples. Early results suggest that learned approaches can outperform classical approaches significantly and provide orders of magnitude performance improvement when integrated into DBMSs.

Instance-specific learning. The predominant approach today for learning database components is *instance-specific learning*. The idea of instance-specific learning is to execute a representative workload (i.e., a set of SQL queries) over a given database[1] and then use the observations to train ML models to solve database tasks. For example, for a learned query optimizer, [9, 12–14] a large set of SQL queries needs to be executed to collect query plans and their runtimes, which then serve as training data for learning an ML model that can determine an optimal query plan (i.e., a plan with minimal runtime) for a given SQL query. The rationale is to exploit instance-specific information to overcome the failures made by traditional approaches using generic heuristics and simplifying assumptions. However, training one-off models comes at the cost of high training overheads and the cost of maintaining many models.

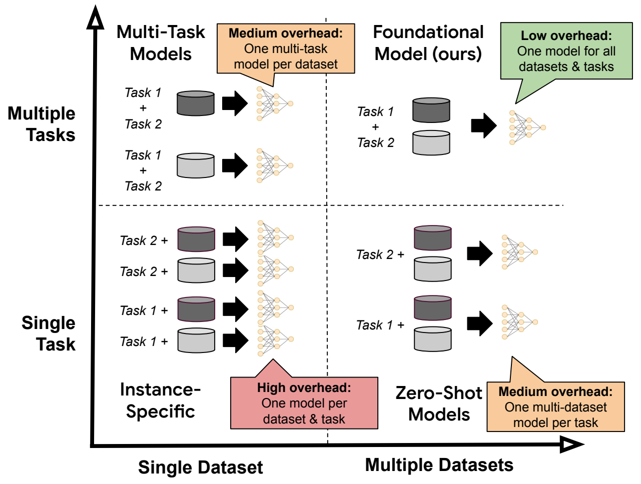
Multi-Task Models
Medium overhead: One multi-task model per dataset
Foundational Model (ours)
Low overhead: One model for all datasets & tasks
Multiple Tasks
Task 1 + Task 2
Task 1 + Task 2
Task 1 + Task 2
Task 2 +
Task 2 +
Task 1 +
Task 1 +
Task 2 +
Task 1 +
Single Task
Instance-Specific
High overhead: One model per dataset & task
Zero-Shot Models
Medium overhead: One multi-dataset model per task
Single Dataset
Multiple Datasets

Figure 1: Foundation database models can generalize across tasks and datasets which is very different from current approaches where we need to train multiple models either per task (Zero-shot) or per dataset (Multi-Task) or even per combination (Instance-specific).

High overheads for learning. The high cost for training a one-off model result from collecting training data by executing a large number of queries on potentially large databases. For instance, in [13] it was shown that tens of thousands of SQL queries need to be executed to collect training data for learning an accurate query optimizer where running the training queries can take hours or even days. As such, as shown in Figure 1 (left-bottom) for every new dataset and task at hand, a different one-off model needs to be trained from scratch which causes extremely high overhead since for every new dataset and task, training data needs to be collected and a new model instance needs to be trained. Combined with the need to maintain many models, instance-specific approaches have limited adaption in industry.

[1]Throughout this proposal, we use the term database and dataset interchangeably to refer to a particular dataset of tables with certain data characteristics.

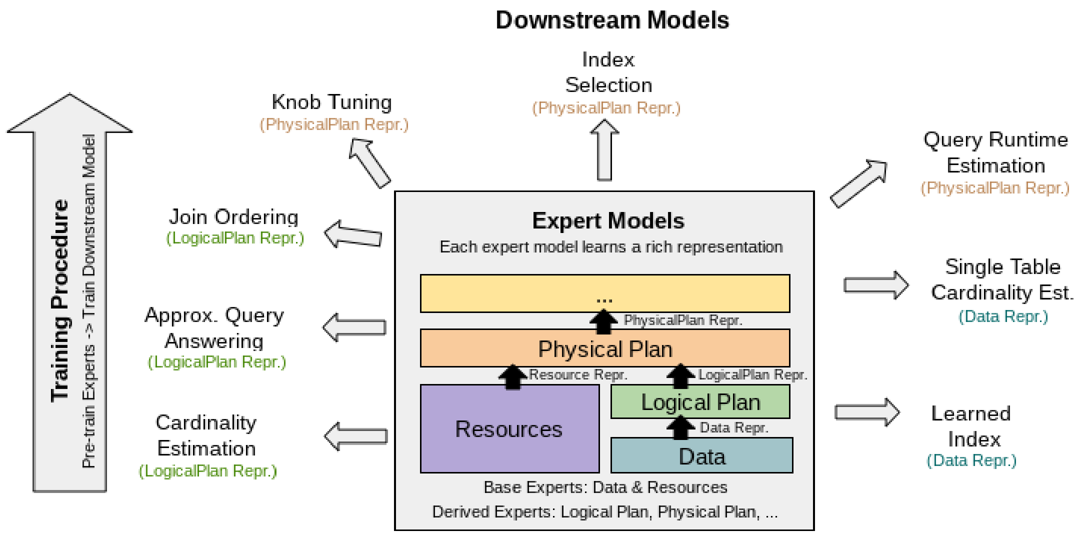

Figure 2: Foundation database models build on a mixture pre-trained experts where some experts learn representations independently (e.g., the data expert) and experts that enrich representations (e.g., the logical plan and physical plan expert). Shallow downstream models take these representation as input and solve a particular database task.

Towards generalizable models. As such, to lower training overheads, new approaches for learning database tasks have been proposed. A recent interesting direction is multi-task learning approaches for learned database components that enable the same model to be used across several database tasks by using shared knowledge. For example, [21] has presented a multi-task model that can be used for cardinality estimation and cost estimation. While these models can lower the training overhead since shared parts that are similar across tasks only need to be trained once, these models still require that one model is trained per dataset (see Figure 1, top-left). Another direction is to use models that can generalize to unseen dataset (see Figure 1, right-bottom). For example, [4] presented a zero-shot cost model which is pre-trained across several datasets which can then can be used on unseen datasets out-of-the-box. Unlike multi-task approaches, zero-shot approaches can generalize across datasets, however, on the contrary they still require one model per task.

Foundation database models. In this paper, we propose a different approach to learned database tasks which we call foundation database models. The key difference to existing learning approaches is that foundation database models can generalize across both tasks and datasets (see Figure 1, top-right) and thus reduce training overhead significantly. Recent advances in LLMs have proven that generalizable (i.e., foundation) models for text, coupled with fine-tuning, can be used to solve a wide variety of NLP problems[16]. In this paper, we suggest a similar approach for database tasks and thus call this model a foundation model for databases.

The insight to realize such a foundation model for database tasks is that we use a mixture of pre-trained expert models as shown in Figure 2 which enable generalizability along the two dimensions: (1) To generalize across datasets, we provide a data expert that learns to *summarize* databases into learned embeddings which represent the characteristics of a given dataset. The data expert does not use database specific information such as particular constants in the data or attribute and table names to learn the embedding, but it uses a new transferable-encoding of databases which allows the data expert to learn general data characteristics of a database such as data distributions and correlations. (2) Moreover, foundation database models come with additional pre-trained experts that enrich the data expert to solve a wide range of downstream tasks with only low overheads. For example, as shown in Figure 2 the learned data representation of an expert for data summarization can be enriched by logical plan expert which captures the knowledge of how relational algebra transforms data. The logical plan representation as such enriches the data representation and can be used to solve tasks such as cardinality estimation or even approximate query answering where this information is needed. (3) Finally, there might be many more dimensions that foundation models should generalize beyond datasets and tasks. One example is that databases are deployed on different hardware resources which typically require that models (e.g., a learned cost model) need to be retrained per hardware platform. As we discuss in the next section, our framework of using pre-trained experts allows the easy addition of new experts to enable generalization across more dimensions.

Contributions and Outline. As the main contribution of this paper, we present the vision of foundation database models. We present an implementation of our proposed model architecture, and present initial results that show the feasibility and efficacy of this idea. The remainder of the paper is structured as follows: Section 2 outlines our model architecture to enable foundation database models. Afterwards, we present a feasibility study with a first prototype for such a model in Section 3, showing promising results on various tasks that require different inputs. Finally, we discuss the research roadmap in Section 4.

2 The Vision: Foundation Database Models

In this section, we discuss the key ideas that enable a foundation database model that can generalize across tasks, datasets and beyond.

2.1 Expert Models

At the core of our foundation model for database tasks are various expert models that are pre-trained to learn representations of data, logical and physical query plans and beyond that can be used to solve a wide spectrum of downstream tasks. The basic idea is that each expert learns rich representations (i.e., vectors in high dimensional embedding spaces) that capture relevant information for individual aspects relevant for different database tasks. An overview of expert models, the representations they learn, as well as the downstream tasks enabled by the representations is shown in Figure 2. Important is that the overview shown in Figure 2 is a framework where additional experts and representations can be added to support generalization across more dimensions and thus even more downstream tasks.

For all the expert models, we differentiate between two types of experts. (1) The first type are base experts that learn representations independently of other experts (see bottom layer in Figure 2). These base expert models can be used standalone to support a certain downstream task or enriched by other (dependent) experts. One example for a base expert is the data expert model, that learns to summarize tabular data using learned data representations. Another example for a base expert is a resource expert, which learns to summarize hardware characteristics of the infrastructure a database is running on. (2) The second type of experts, which we call dependent experts, are models that learn to enrich representations from base experts (see upper layers in Figure 2). These dependent experts use representations of other experts as input and add more information to their representations. For example, the logical plan expert produces representations that capture the effects of logical query plans when applied to a set of tables. As such it uses the table representations as input and produces an output vector that represents the data characteristics of a query result which then can be used for tasks such as cardinality estimation or approximate query answering. Another example is a physical expert which builds on the resource and the logical plan expert and adds information about (algorithmic) query complexity to the representations of logical plans and resources. For example, the physical plan representation jointly capture effects of algorithms (e.g., hash vs. nested loop join) and hardware (e.g., memory speeds and size) which are important for downstream tasks such as runtime estimation or knob tuning.

2.2 Pre-Training of Experts

An important aspect of expert models is that once they are pre-trained, they then can be used out-of-the-box or with only minimal fine-tuning to solve a new downstream task (e.g., cardinality estimation) on a new dataset. Important is that the pre-training captures the relevant information which allows the experts to generalize while experts have different roles.

A core expert to enable the generalizability of foundation models across datasets is the data expert. During pre-training, the data expert model learns to capture important information like table sizes and distribution of data in a table in a succinct *database-independent* manner. That way, once pre-trained, the data expert can be used to summarize unseen datasets as a vector. As we discuss later in Section 3, to learn such a data representation, we encode the data of tables using a new relative encoding which allows the data expert to capture the nature of data distributions (e.g., uniform vs. skewed) in a dataset independent manner. During pre-training, we use a set of novel pre-training tasks which capture data characteristics along rows and columns (cf. Section 3). In future, the data expert can be extended even further to also capture aspects of physical data layouts such as sorting keys and partitioning information.

Similarly, the other experts are also pre-trained such that they capture relevant information for producing rich output representations. It is important that during pre-training the individual experts learn the intrinsic characteristics of the component they represent in a *task-independent* manner with particular designed pre-training task. This allows the learned representations to be useful across a wide spectrum of downstream tasks. For example, as mentioned before, the physical plan expert learns the inherent characteristics of physical query plans (i.e., the complexity of the physical operators in a plan). For pre-training such an expert, as we discuss in Section 3, we use simple query cost models from traditional database systems which capture effects of runtime and I/O for physical plans and train the expert such that we can predict these cost from the learned physical plan representation. As we see later, this representation can then be used for various tasks which need to understand query complexity such as runtime estimation but also other cost-related tasks such as index selection.

2.3 Realizing Downstream Tasks

Based on the pre-trained expert models, we can then realize models for downstream tasks which use the learned representations of the experts as input. As representations are rich, it often suffices to use a simple, shallow model to implement a downstream task as we will show in our feasibility study in Section 3. For instance, it suffices to use a simple shallow MLP regression model which uses the learned logical plan representation as input to solve the cardinality estimation task. Similarly, for a runtime estimation model, we could use another regression MLP which uses the physical plan representation as input and predicts runtime as a regression task. However, the output representations of the pre-trained experts can also be coupled with more involved model architectures to solve more complex problems such as knob tuning. Figure 2 shows a collection of possible downstream tasks and their required input representations. An important aspect is that building upon the pre-trained experts reduces the need for task-specific training data for a downstream model, which is much less than learning the task from scratch. Furthermore, once a downstream model (e.g., for cardinality or runtime estimation) is trained, *it can be reused and do not need to be retrained for different databases*. This is due to the nature of foundation models where the data expert captures information about tabular data in a database-independent manner.

3 Feasibility Study: A First Prototype

In this section, we present a feasibility study of implementing a first foundation database model. For our initial validation, we developed three expert models: the data expert to enable generalizability across datasets, as well as a logical plan expert and a physical plan expert to support various different downstream tasks that are based on representations of these experts. Further experts such as a resource experts which allows foundational models to generalize across

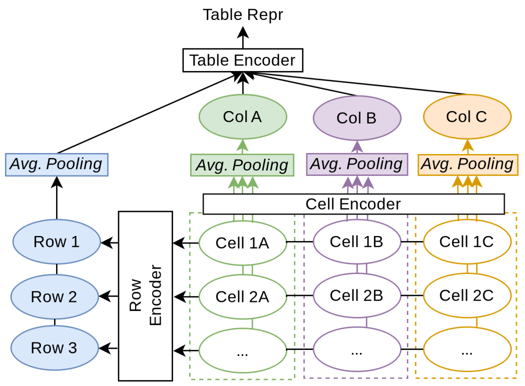

Figure 3: The data expert model aggregates data characteristics of tabular data at different levels. The data experts focuses on learning statistical and distribution information of tabular data and thus use different features including statistical information such as table sizes and number of unique values per column as well as novel row- and column-wise pre-train objectives to aggregate data characteristics.

different hardware or other experts are interesting avenues of future work. In the following, we explain the model architectures and pre-training of the three before-mentioned experts as well as how we realized downstream tasks using these experts. At the end of the section, we present results of an initial evaluation.

3.1 Model Architecture

We next provide details of the model architectures used for the different expert models in our prototype.

Data Expert: The task of the data expert model is to learn representations of tabular data that capture important data characteristics for a wide spectrum of possible database tasks that are either supported directly on the output of the data expert or by further processing the representation by other expert models as we describe in the next section. For this, the data expert model (shown in Figure 3) captures data characteristics at the cell levels and aggregates the information along rows and columns to capture important column statistics (e.g., data distributions within a column), as well as value co-occurrences in rows. For computing a representation of a table, we sample a variable number of rows from the base table and featurize them cell by cell. We featurize cells by normalizing to values between 0 and 1 [10]. To encode categorical and string values, we first assign numeric identifiers in lexicographical order before applying the normalization mentioned before.

Such normalized featurization is needed to learn representations that are not tied to actual values in a particular database. In addition, higher level statistics such as number of rows in the table, unique values per column, the NULL-ratio, etc. are added as input features to the data expert model. As such, two tables from different domains but with the same sizes and data distributions (e.g., all uniform) will result in similar representations. The representation thus summarize important data characteristics in a dataset-independent manner, which allows us to use the data expert on unseen datasets as we show in our experimental evaluation in Section 3.3.

To compute the table representation from the data of a given table , we first encode cell values and features by a 4-layer MLP

	Q-error (Median)	Q-error (95th)
1-dimensional	1.89	78.68
2-5 dimensional	2.32	193.85
Overall	1.97	90.06

Table 1: Accuracy of the column-wise pre-training task of histogram reconstruction to learn data distributions. Results are averaged over all 19 datasets.

Correlation of Data	Accuracy (validation data)	Accuracy (unseen dataset)
High	93%	80%
Low	91%	77%

Table 2: Accuracy of the row-wise pre-training task. We report results split into high correlation rows ($Pearson > 0.8$ between input and predicted row value) and low correlation ($Pearson|Spearman|RDC > 0.75$)). Higher correlation values are easier to reconstruct.

per cell as shown in Figure 3. We use these cell representations to construct row and column representation. For row representations, we append all cell representations of a row into a vector and use another 4-layer MLP to compute a row representation (blue nodes in Figure 3). The row-representation for the rows in the data sample of a table are then passed through a row aggregator which summarizes characteristics across rows (i.e., avg-pooling on top of rows). Similarly, we use avg-pooling on top of a column resulting in a column representation. The learned row embedding and the column embeddings are then passed to a final MLP producing the final table representation. To achieve this, we use an architecture with 3 MLPs (4 layers each).

Discussion. We want to call out that our data expert is different than existing line of work on table representation learning such as TURL [1]. These earlier approaches for table representation learning aim for a semantic understanding of the table data to solve tasks such as semantic type annotation or entity linking. However solving performance related learning problems in databases, such as cardinality estimation or knob tuning, such semantic understanding of the table data is in most cases not really necessary. Instead, our models focus on learning statistical and distribution information of tabular data and thus use different features including statistical information such as table sizes and number of unique values per column. Moreover, we use pre-train objectives for our data expert which are different from pre-training objectives used by existing table representation learning approaches as we discuss below.

Logical Plan Expert: A logical plan can be naturally described by a directed acyclic graph (DAG) where leaf nodes use representations of the data expert as input. To that end, we use graph neural networks (GNNs) to learn representations of logical query plans as shown in Figure 4 (blue area). We use a heterogeneous graph where each node represents a logical plan operator, and there are as many node types as the different plan operators (tables, filters, joins etc.). The table node types are initialized by the data expert model. We define details of each operator as node features. For example, a filter operator has features like comparison operator (>, >= etc.) and filter literal where the literal is encoded with normalized, relative

Raw Value	Relative Encoding
age > 40	age > 0.3
name = 'Max'	name = 0.7
price > 100	price > 0.1

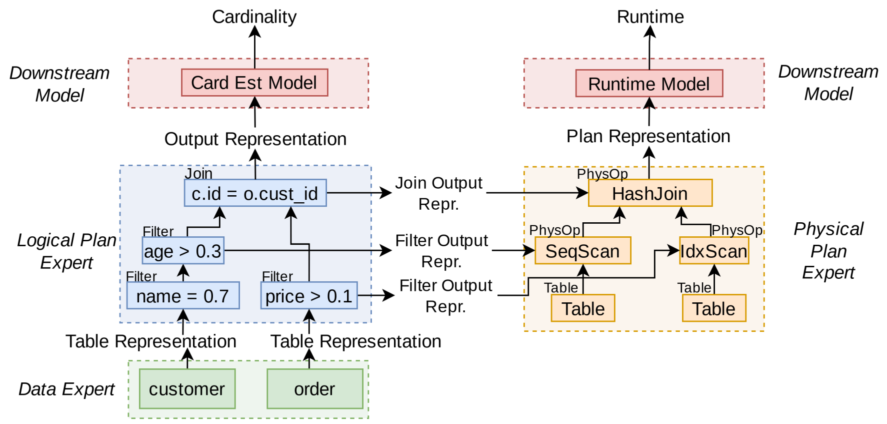

Figure 4: The logical plan expert (blue) is layered on top of the data expert (green). The physical plan expert (orange) is layered on top of the logical expert (blue). The experts uses a relative encoding of data and predicate constants as shown on the left. Downstream models (red) predict cardinalities or runtime of query plans based on the output of the two plan experts.

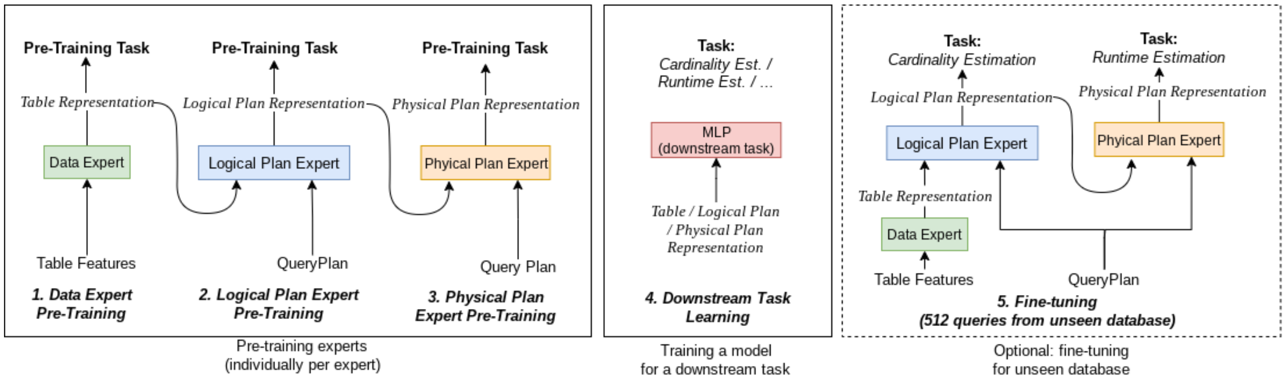
Pre-Training Task
Pre-Training Task
Pre-Training Task
Table Representation
Logical Plan Representation
Physical Plan Representation
Data Expert
Logical Plan Expert
Physical Plan Expert
Table Features
QueryPlan
Query Plan
1. Data Expert Pre-Training
2. Logical Plan Expert Pre-Training
3. Physical Plan Expert Pre-Training
Pre-training experts (individually per expert)
Task: Cardinality Est. / Runtime Est. / ...
MLP (downstream task)
Table / Logical Plan / Physical Plan Representation
4. Downstream Task Learning
Training a model for a downstream task
Task: Cardinality Estimation
Task: Runtime Estimation
Logical Plan Representation
Physical Plan Representation
Logical Plan Expert
Physical Plan Expert
Table Representation
Data Expert
Table Features
QueryPlan
5. Fine-tuning (512 queries from unseen database)
Optional: fine-tuning for unseen database

Figure 5: Training Steps. 1. Pre-Training the base experts (e.g. Data Expert). 2 & 3: Pre-train derived experts by making use of preceding experts. 4. Task-specific downstream models are learned. Steps 1 to 4 are performed on 18 datasets (different from our target database). 5. Fine-tune the experts and the downstream model on the target database is optional.

values. The GNN topological message passing follows the direction of the logical plan graph (from leaf nodes to a single root). In every iteration of message passing, a GNN-MLP (Multi-Layer-Perceptron) is used to combine the incoming message with the node's hidden state. This staging of intermediate results of applying each operator essentially models the filter and join operations over the original data. The hidden state of the single root node is finally read and passed to the downstream tasks. We use a GNN-MLP with 6 layers.
Physical Plan Expert: For the physical plan expert model, we extend the GNN architecture of the logical plan expert. The nodes of the physical plan expert are physical operators, and edges correspond to the data flow between them as shown in Figure 4 (orange area). Each node in the logical plan graph has a corresponding node in the physical plan graph describing how the logical operator is executed in terms of physical operators. Both nodes are connected via an edge. The learned representation of the logical plan node will be passed to the corresponding physical plan node during message passing. Figure 4 shows the correspondence between the logical plan graph (blue) and physical plan graph (orange). Notably, the embedding vectors for intermediate results from the logical plan are passed along the connecting edges to the corresponding nodes in the dependent physical plan expert. This is an important step, since these embeddings capture data characteristics of intermediate results such as cardinalities.

3.2 Training Procedure and Details

Our training methodology is shown in Figure 5. The initial phase involves pre-training of our expert models (step 1-3) which needs to be done once on a large training corpora. Afterwards, downstream models are realized on a per-task basis with only minimal training data (step 4). Optionally, fine-tuning can be used to further improve accuracy (step 5).
Pre-training. The data expert is pre-trained first (step 1 in Figure 5) in order to learn table representations. For the data expert, we apply pre-training on a disjoint set of databases not used for the evaluation of the downstream tasks to show that our models enable the generalization to unseen databases. We use the database corpora provided in [4, 15] and split the corpus such that 18 databases are

used for pre-training and the 19th database is used for evaluation[2]. We pre-train the data expert with 1000 row samples per table. We use two pre-training tasks (1) column-wise distribution learning and (2) masked-out row value reconstruction. The first pre-train tasks to learn column-wise distributions is to restore histogram information (i.e., bucket counts) for up to 1-5 dimensional histograms from column embeddings, while the second task restores masked out row values from row embeddings. The results of pre-training the data expert can be seen in Tables 1 and 2.

The logical plan expert pre-training (step 2 in Figure 5) takes as input the learned data representation and a logical query plan to construct the GNN for the logical plan expert. To pre-train the GNN-based model of the logical plan expert in our prototype, we executed in total 150k SQL queries uniformly distributed over 18 data sets and collected the logical plans and the query results for those queries. For the pre-training, we use the same tasks as for the data expert but apply them to the logical query plan representation which represents the query result. In the current setup of our case study, the pre-training dataset contains queries with up to 2 joins and 5 filters, following the observation of [20] that most queries in analytical workloads come with less than 2 joins. However, in future we plan to scale up the logical plan expert to more joins or explore other directions such as how fine-tuning of the experts (see below) can be used to expand to more complex workloads while using only simple workloads for pre-training.

Finally, the physical plan expert is pre-trained using learned representations of the logical plan as input to construct the GNN for the physical plan expert (step 3 in Figure 5). For the pre-training of the physical expert, we use the physical plans of the same 150k SQL queries over the 18 data sets we use for the pre-training of the logical plan expert. As pre-traning task, we learn to predict query cost coming from a traditional database cost model from the representation of the physical expert.

Realizing downstream models. Based on the the learned representations of the experts from the pre-training phase, models for different downstream tasks can be realized (step 4-5 in Figure 5). It is important that the learned representations of step 1-3 support to realize multiple tasks with low overhead. For our case study, we trained only simple MLPs for the downstream tasks. The initial evidence for this is shown in our initial evaluation at the end of this section. In future, we aim to show that a foundation database model which uses an architecture as discussed above can generalize across many more tasks.

Fine-tuning. Finally, we want to emphasize that we can use fine-tuning to improve the accuracy of the experts and a task-specific downstream model on a new unseen dataset (step 6 in Figure 5) in an end-to-end manner. In our initial evaluation, we fine-tune experts and downstream models on an unseen database with 512 additional queries, showing the efficiency of our pre-training. For a comparison, one-off models need 100k queries per database to learn a new unseen task on a given database, making fine-tuning with a few hundred queries a highly appealing option considering the small amount of training data that is needed.

[2]IMDB has been left out due to licensing issues of the data set which disallowed the use inside Google.

	Ours	**Ours (FT)**	**PG**	**DeepDB**	**MSCN**
Q-error (Median)	2.12	1.69	1.98	1.83	1.68
Q-error (95th)	92.92	26.08	294.15	152.23	3120

Table 3: Cardinality estimation using the logical plan expert. The workload is composed of query plans with up to 2 joins and up to 5 filter predicates. Results are averaged with leave-one-out cross validation over all 19 unseen datasets. Fine-tuning (FT) has seen 512 queries on the target database.

3.3 Initial Evaluation

In the following, we show the results of our initial evaluation where we use the pre-trained experts, as explained before, and learn three different downstream task models based on the representations.

Task 1: Cardinality Estimation with Data & Logical Expert. First, we evaluate on the task of cardinality estimation using a regression MLP which takes the logical plan representation as input which builds on the data expert. For learning the task, we use a workload comprised of query plans with up to two joins and filters with up to five predicates. We tested the learned cardinality estimator with and without fine-tuning with 512 additional queries on the unseen datasets. As our results in Table 3 show, our foundation model (i.e., the pre-trained experts with the downstream model for cardinality estimation) can outperform Postgres and achieves competitive performance w.r.t DeepDB [6] and MSCN [7], which are one-off models that need to be trained per task and database.

Task 2: AQP using the same Data and Logical Expert. To show task-independence of our logical plan representation, we use the the same representation as for the previous task but now for approximate query answering which is implemented as a second MLP on top of the same representation. We predict results of average, and min/max aggregation queries with 1-5 filter predicates and up to two joins. Again, we achieve high accuracy (i.e., a median q-error of 1.27 for avg and 1.31 for min/max) on an unseen database without fine-tuning.

Task 3: Runtime Estimation by adding the Physical Expert. Finally, as a last task we implement runtime estimation as a MLP on top of the physical plan expert to show that through stacking of experts we can enrich embeddings and thus enable additional tasks. The q-error for this downstream task is shown in Table 4. We compare to a state of the art learned cost model which uses a task-specific but zero-shot architecture that can generalize across databases (ZS Cost [4]), as well as a Postgres-based cost estimator. As we can see, our foundation model, which is task-independent provides high accuracy out-of-the-box and is highly competitive with the task-specific ZS model in particular when our model is fine-tuned on the unseen data set with a few queries only.

4 Research Roadmap

While the feasibility study in Section 3 shows promising results, this paper is just a starting point. To deliver on this vision, with this paper we aim to foster research in this direction.

Further tasks and experts. While our feasibility study showcases a foundation model based on three expert models that can already solve different core database tasks, we believe that a much wider

	Ours	**Ours (FT)**	**ZS Cost**	**PG Cost**
Q-error (Median)	1.87	1.5	1.08	6.44
Q-error (95th)	10.76	6.83	1.7	19.73

Table 4: Runtime estimation using the physical plan expert. Workload is composed of query plans with 0-2 joins and 0-5 filter predicates. Results are averaged with leave-one-out cross validation over all 19 unseen datasets. Fine-tuning (FT) has seen 512 query plans on the target dataset.

spectrum of database tasks can be supported by the architecture proposed in this paper. First, even with the three expert models many more tasks can be realized. Moreover, when adding more expert models, existing downstream tasks can either be extended or more tasks can be supported. For example, when adding an expert model that understands effects of different hardware resources (CPU, memory/disk, network) as shown in Figure 2 one could enable run-time estimation models that take effects of different hardware into account. Furthermore, when adding other experts such as an expert for SQL query strings, new tasks such as query rewrite on the query string level or query completion can be realized.

Deploying in the cloud. Foundation database models eliminate the high training cost of one-off models as discussed before. Although this is a huge opportunity for wide adaption in the cloud, there are still many open research challenges involved with foundation database models. For example, an open question is how much pre-training these models require to generalize robustly to a wide spectrum of tasks. Moreover, foundation models bring other benefits beyond reducing the high efforts that come with one-off models. For example, an often ignored problem of cloud providers is that they are not allowed to use the customer databases to collect training data by running queries, due to strict security regulations that cloud providers impose. Hence, foundation models might be extremely attractive to address this problem since they can be pre-trained on available non-customer databases (e.g., public data), and then be used out-of-the-box or with minimal fine-tuning on a unseen customer database.

Beyond database systems. Finally, we believe that our approach of using composable expert models that learn representations of individual aspects are a much generic paradigm for learned systems tasks and can be used beyond database systems. For example, cloud resource management problems of non classical database workloads (e.g., ML training or inference) also need to understand workload characteristics as well as available hardware resources to make an optimal decision. Leveraging insights from our work, this learned optimization may also be broken down into a workload expert and a hardware expert. Each expert can be trained independently with large amounts of data that capture complexity and diversity in cloud workloads or hardware configurations. These experts will then be combined with different downstream models to jointly optimize goals such as reducing application run time or improving server resource utilization.

Acknowledgments

We thank Yannis Chronis (Google SRG) and Sami Abu-El-Haija (Google Research) for their support and discussions on this paper.

References

[1] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *arXiv:2006.14806 [cs]* (Dec. 2020).

[2] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-Dimensional Index for Correlated Data and Skewed Workloads. *Proc. VLDB Endow.* 14, 2 (Oct. 2020), 74–86.

[3] Roman Heinrich, Carsten Binnig, Harald Kornmayer, and Manisha Luthra. 2024. Costream: Learned Cost Models for Operator Placement in Edge-Cloud Environments. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024.* IEEE, 96–109.

[4] 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.

[5] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. 2020. Learning a Partitioning Advisor for Cloud Databases. In *Proc. ACM SIGMOD (SIGMOD '20).* Association for Computing Machinery, New York, NY, USA, 143–157.

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

[7] 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.*

[8] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proc. ACM SIGMOD (SIGMOD '18).* ACM, New York, NY, USA, 489–504.

[9] S. Krishnan, Z. Yang, Ken Goldberg, J. Hellerstein, and I. Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *ArXiv* abs/1808.03196 (2018).

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

[11] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19).* Association for Computing Machinery, New York, NY, USA, 270–288.

[12] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proc. ACM SIGMOD (SIGMOD/PODS '21).* Association for Computing Machinery, New York, NY, USA, 1275–1288.

[13] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (July 2019), 1705–1718. https://doi.org/10.14778/3342263.3342644

[14] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM'18).* Association for Computing Machinery, New York, NY, USA, Article 3, 4 pages.

[15] Jan Motl and Oliver Schulte. 2015. The CTU Prague Relational Learning Repository. *CoRR* abs/1511.03086 (2015).

[16] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746.

[17] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *Proc. ACM SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 985–1000.

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

[19] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-Based Cost Estimator. *Proc. VLDB Endow.* 13, 3 (Nov. 2019), 307–319.

[20] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC Is Not Enough: An Analysis of the Amazon Redshift Fleet. *Proc. VLDB Endow.* 17, 11 (2024), 3694–3706.

[21] Ziniu Wu, Pei Yu, Peilun Yang, Rong Zhu, Yuxing Han, Yaliang Li, Defu Lian, Kai Zeng, and Jingren Zhou. 2022. A Unified Transferable Model for ML-Enhanced DBMS. In *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022.* www.cidrdb.org.