NeuroDB: Efficient, Privacy-Preserving and Robust Query Answering with Neural Networks

Sepanta Zeighami
University of Southern California
zeighami@usc.edu

Cyrus Shahabi
University of Southern California
shahabi@usc.edu

Abstract

The Neural Database framework, or NeuroDB for short, is a novel means of query answering using neural networks. It utilizes neural networks as a means of data storage by training neural networks to directly answer queries. That is, neural networks are trained to take queries as input and output query answer estimates. In doing so, relational tables are represented by neural network weights and are queried through a model forward pass. NeuroDB has shown significant practical advantages in (1) approximate query processing, (2) privacy-preserving query answering, and (3) querying incomplete datasets. The success of the NeuroDB framework can be attributed to the approach learning patterns present in the query answers, utilized to learn a compact representation of the dataset with respect to the queries. This allows learning small neural networks that accurately and efficiently represent query answers. Meanwhile, learning such patterns allows for improving the accuracy in the presence of error, with such robustness to noise allowing for improved accuracy in the case of private query answering and query answering on incomplete datasets. This paper presents an overview of the NeuroDB framework and its applications to the three aforementioned scenarios.

1 Introduction

The Neural Database framework, or NeuroDB for short, is a novel means of query answering using neural networks [33, 31, 32]. Instead of iterating over the records in a dataset to answer queries, it trains neural networks to predict query answers. The neural networks are trained in a supervised learning fashion, where the model takes a query as input and outputs an estimate of the query answer. Datasets are represented by neural network weights and are queried through a model forward pass. NeuroDB has shown significant practical and theoretical advantages in three settings: (1) approximate query processing [33], (2) privacy-preserving query answering [31], and (3) querying incomplete datasets [32]. In this paper, we present an overview of the NeuroDB framework and its applications to the aforementioned scenarios, summarizing the results in [33, 31, 32].

NeruoDB follows a function approximation view of database operations. For a database D, define a *query function* $f_D(q)$ as a function that takes a query, q, as an input and outputs its correct query answer. NeuroDB trains a neural network, $\hat{f}(q;\theta)$, that takes a query q as its input and outputs an estimate to the query answer. The training objective is to ensure $\hat{f}$ and f_D are similar, e.g., $\sum_{q\in Q} |\hat{f}(q;\theta) - f_D(q)|$ is minimized for a set of possible queries Q, so that query answer estimates are accurate. Following this framework, to apply NeuroDB to a specific application scenario, we need to specify a query representation, and a training procedure for the application. NeuroDB inherently produces query answer estimates, not exact answers, and thus applies to scenarios where exact answers are not required. This is true in the three scenarios mentioned above.

In the approximate query processing scenario, the goal is to provide efficient and approximate query answers. We specifically consider answering range aggregate queries (RAQs), which are the building block of many real-world applications (e.g., calculating net profit for a period from sales records. RAQs filter a dataset by range predicate and ask for an aggregation of some attribute in the filtered

Table Representation Learning Workshop at NeurIPS 2023

dataset. Due to the large volume of data, exact answers can take too long to compute and fast approximate answers may be preferred. There is a time/space/accuracy trade-off, where algorithms can sacrifice accuracy for time or space. The results in [33] show that applying the NeuroDB framework can provide better time/space/accuracy trade-offs than existing methods. Furthermore, [33] theoretically analyzes the approach, showing novel theoretical characteristics in using a neural network to perform database queries. Specifically, [33] theoretically shows how the accuracy of the neural network can depend on data and query characteristics.

In the privacy-preserving query answering scenario, the goal is to answer queries on a database while preserving the privacy of the users who contributed records to the dataset. Specifically, [31] considers answering `COUNT` queries on location datasets, that is, answering how many people are in a certain location. Such queries are common on location datasets, often collected from mobile apps and used for various purposes such as optimizing traffic or studying disease spread. Differential privacy is often used to protect privacy and ensure that the location of a specific user cannot be inferred when releasing aggregate location information from such datasets. [31] shows how the NeuroDB framework can be applied to this setting by training a model while preserving differential privacy to answer the queries. Results in [31] show that using the NeuroDB framework, one can answer queries much more accurately than existing methods, while providing the same privacy guarantees.

Finally, in the query answering on incomplete datasets scenario, the goal is to provide accurate query answers while the observed dataset contains missing records. Real-world databases are often incomplete for various reasons, including the cost of data collection, privacy considerations or as a side effect of data integration/preparation. For instance, to know housing prices in an area, collecting information for every house is costly, if not impossible, but Airbnb already provides a sample for free [1] (dataset is a sample because it only contains Airbnb prices and not other housing sources). In such scenarios, some records are entirely missing from the datasets. Meanwhile, OLAP applications require answering aggregate queries on such incomplete datasets. [32] studies this problem, aiming to provide accurate query answers while only having access to such an incomplete dataset. The results in [32] show that the NeuroDB framework can provide much more accurate query answers than existing methods. The challenge in such a scenario is to train a model using the incomplete dataset that generalizes well to the complete dataset. This requires carefully designed query representation and training procedures, as done in [32].

The success of the NeuroDB framework in the above scenarios can be attributed to the approach learning patterns present in the query answers. Patterns in the query answers can be utilized to learn a compact representation of the dataset with respect to the query. This allows learning small neural networks that accurately and efficiently represent a query function. Meanwhile, learning such patterns allows for improving the accuracy in the presence of error, with such robustness to noise allowing for improved accuracy in the case of private query answering as well as query answering on incomplete datasets. NeuroDB generalizes the recent trend of replacing different database components with learned models [10, 13, 20, 18, 17, 27], where the main observation has been that a certain database operation (e.g. retrieving a record's location in indexing [14]) can be replaced by a learned model. NeuroDB follows the overarching idea that answering the query itself can be performed by a model, since query answering is a function that can be approximated.

The rest of this paper describes the application of the NeuroDB framework to each of the scenarios discussed above. Sec. 2 presents results for approximate query processing, Sec. 3 presents results for privacy-preserving query answering, Sec. 4 presents results for querying incomplete dataset and Sec. 5 concludes the paper.

2 Answering Range Aggregat Queries

In this section, we provide an overview of [33], which shows an application of NeuroDB for answering range aggregate queries. [33] also theoretically analyzes the approach, which we omit due to space.

2.1 Preliminaries

Problem Definition. Consider the following SQL query on a dataset D with n records and $\bar{d}$ attributes, A_1, ..., $A_{\bar{d}}$. Assume, w.l.o.g, that the attribute values are in the range $[0, 1]$ (or they can otherwise be scaled).

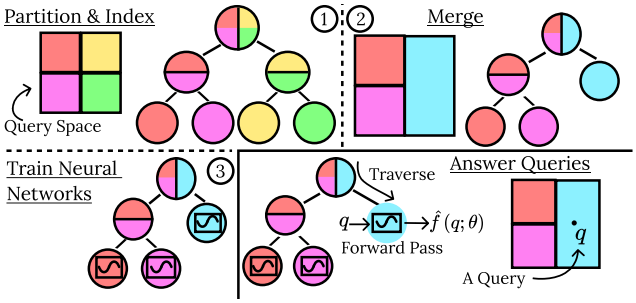

Figure 1: NeuroSketch Overview

$$\texttt{SELECT AGG}(A_m)\ \texttt{FROM}\ D\ \texttt{WHERE}$$
$$c_1 \leq A_1 < c_1 + r_1\ \ \texttt{AND}\ \ \ldots\ \ \texttt{AND}\ \ c_{\bar{d}} \leq A_{\bar{d}} < c_{\bar{d}} + r_{\bar{d}}$$

For any i, c_i and $c_i + r_i$ are the lower and upper bounds on the attribute A_i. c_i and r_i can be 0 and 1, respectively, in which case there are no restrictions on the values of A_i in the query. `AGG` is a user defined *aggregation function*, with examples including `SUM`, `AVG` and `COUNT` aggregation functions. A_m is called the *measure attribute*, where m is an integer between 1 and $\bar{d}$. Let $c = (c_1, ..., c_{\bar{d}})$ and $r = (r_1, ..., r_{\bar{d}})$ be $\bar{d}$-dimensional vectors. We call the pair $q = (c, r)$ a *query instance*. Different query instances correspond to different range predicates for the measure attribute A_m and aggregation function `AGG`. We define the *query function* $f_D(.)$ so that for a query q, $f_D(q)$ is the answer to the above SQL statement. Furthermore, define $\mathcal{Q} = \{(c, r) \in [0, 1]^d, c_i + r_i \leq 1 \forall i\}$ as the set of all possible queries. The problem studied in this section is to answer the queries in $\mathcal{Q}$ efficiently and accurately, that is, given an accuracy requirement, answer queries as fast as possible while providing the desired accuracy level.

Related Works. This problem has been extensively studied in the approximate query processing (AQP) literature. Existing methods can be divided into sampling-based methods [11, 3, 5, 22] and model-based methods [7, 26, 17, 27, 13]. Sampling-based methods use different sampling strategies (e.g., uniform sampling, [11], stratified sampling [5, 22]) and answer the queries based on the samples. Model-based methods develop a model of the data that is used to answer queries. The models can be of the form of histograms, wavelets, data sketches (see [7] for a survey) or regression and density based models [17, 27, 13]. Generally, these works follow two steps: first, a model of the data is created, and second, a method is proposed to use these data models to answer the queries. The results in [17, 27, 13] show that learned methods outperform other existing approaches.

2.2 NeuroSketch Overview

Rather than creating models of the data, as done in the related work [17, 27, 13], we learn neural networks to directly estimate query answers. This helps improve performance as small models are trained to perform a specific task. Specifically, in NeuroSketch [33], the goal is to learn neural networks, $\hat{f}(.;\theta)$, to approximate the query function, $f_D(.)$. Such neural networks take as input an RAQ, q. A model forward pass outputs an answer, $\hat{f}(q;\theta)$. The goal is to train neural networks so that its answer to the query, $\hat{f}(q;\theta)$, is similar to the ground-truth, $f_D(q)$. If such neural networks are small and can be evaluated fast, we can use them to directly answer the RAQ efficiently and accurately, by performing a model forward pass.

The key idea behind NeuroSketch design is that, even on the same database, some queries can be more difficult to answer than others. By allocating more model capacity to queries that are more difficult, we can improve the performance. We do so by partitioning the query space and training independent neural networks for each partition. The partitioning allows diverting model capacity to harder queries. By creating models specialized for a specific part of the query space, *query specialization* allows us to control how model capacity is used across query space.

Fig. 1 shows an overview of NeuroSketch. During a pre-precessing step, (1) we partition and index the query space using a kd-tree. The partitioning is done based on our query specialization principle, with the goal of training a specialized neural network for different parts of the query space. (2) To account for the complexity of the underlying function in our partitioning, we merge the nodes of the kd-tree that are *easier* to answer, so that our model only has to specialize for the certain parts of the

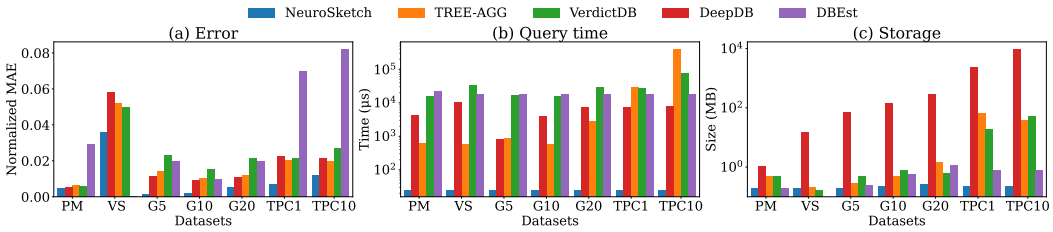

Figure 2: RAQs on different datasets

space that are estimated to be more difficult. (3) After some nodes of the kd-tree have been merged, we train a neural network for all the remaining leaves of the kd-tree. Finally, to answer queries at query time, we traverse the kd-tree to find the leaf node a query falls inside, and perform a forward pass of the neural network. We describe each step next.

Partitioning & Indexing. To partition the space, we choose partitions that are smaller where the queries are more frequent and larger where they are less frequent. This allows us to divert more model capacity to more frequent queries, thereby boosting their accuracy if workload information is available. We achieve this by partitioning the space such that all partitions are equally probable. To do so, we build a kd-tree on our query set, Q, where the split points in the kd-tree can be considered as estimates of the median of the workload distribution (conditioned on the current path from the root) along one of its dimensions.

Merging. We merge some of kd-tree leaves based on some notion of difficulty of query answering. Specifically, we use *Average Query function Change*, AQC, as a proxy for function approximation difficulty, defined as $AQC = \frac{1}{\binom{|Q|}{2}} \sum_{q,q' \in Q} \frac{|f(q)-f(q')|}{\|q-q'\|}$, where $Q \subseteq \mathcal{Q}$ is a set of queries sampled from all possible queries. AQC is defined based on theoretical results in [33] showing that magnitude of function change correlates with function approximation difficulty. This is also empirically verified, where [33] show AQC is correlated with the error of neural networks. To use AQC, we merge leaf nodes of the kd-tree that have the least AQC value, until a desired number of leaves are left.

Training Neural Networks. We train an independent model for each of the remaining leaf nodes after merging. For a leaf node, N, the training process is a typical supervised learning procedure. The ground-truth answer to queries for training can be collected through any known algorithm, where a typical algorithm iterates over the points in the database, pruned by an index, and for a candidate data point checks whether it matches the RAQ predicate or not. This pre-processing step is only performed once to train our model. Once trained, NeuroSketch is much smaller than data and expected to fit in memory, so it will be much faster than disk-based solutions.

Answering Queries. To answer a query, q, first, the kd-tree is traversed to find the leaf node that the query q falls into. The answer to the query is a forward pass of the neural network corresponding to the leaf node.

2.3 Experimental Results

We compare NeuroSketch with other learned methods DBEst [17] and DeepDB [13], which learn a model of the data and use that model to answer queries. This is in contrast to NeuroSketch that learns a model that directly estimates the query answers. We also compare our approach with two sampling based approaches VerdictDB [22] and TREE-AGG, where the latter is a baseline that builds an R-Tree on the sampled data points for faster search. The datasets used are *PM* [16] (contains Fine Particulate Matter measuring air pollution and other statistics), *TPC-DS* [19] (a synthetic benchmark dataset, with scale factors 1 and 10), *Veraset* (which contains anonymized location signals of cell-phones across the US collected by Veraset [2]) and *GMMs* (5, 10 and 20 dimensional Gaussian mixture models with 100 components and random mean and co-variance, referred to as G5, G10 and G20). Experiments presented here answer `AVG` queries with range predicates sampled uniformly at random.

Fig. 2 (a) shows the error on different datasets, where NeuroSketch provides a lower error rate than the baselines. Fig. 2 (b) shows that NeuroSketch achieves this while providing multiple orders of magnitude improvement in query time. Due to NeuroSketch's use of small neural networks, we observe that model inference time for NeuroSketch is very small and in the order of few microseconds, while DeepDB and DBEst answers queries multiple orders of magnitude slower. Furthermore, the R-tree index of TREE-AGG often allows it to perform better than the other baselines, especially for low dimensional data. Finally, Fig. 2 (c) shows the storage overhead of each methods. NeuroSketch

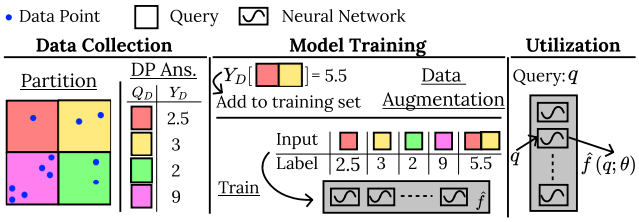

Figure 3: SHN Overview

answers queries accurately by taking less than one MB space, while DeepDB's storage overhead increases with data size, to more than one GB.

3 Answering Spatial Range Count Queries while Preserving Privacy

In this section, we provide an overview of [31] which shows an application of NeuroDB for answering range count queries on location datasets while preserving privacy. [31] contains further discussion on model training and hyperparmeter tuning while preserving privacy, which we omit due to space.

3.1 Preliminaries

Problem Definition. We consider a special case of queries answered by NeuroSketch. We focus on the case when the aggregation function is `COUNT` and the dataset, D consists of users' geo-coordinates (i.e., D is two dimensional consisting of latitude and longitude records). That is, we consider answering range count queries (RCQs) on a spatial database. The goal is to answer an unbounded number of RCQ queries on this dataset while satisfying ε-Differential Privacy [9].

The typical way to solve the problem of answering an unbounded number of RCQs is to design an ε-DP mechanism $\mathcal{M}$ and a function $\hat{f}$ such that *(1)* $\mathcal{M}$ takes as an input the database D and outputs a differentially private representation of the data, θ; and *(2)* the function $\hat{f}(q;\theta)$ takes the representation θ, together with any input query q, and outputs an estimate of $f(q)$. In practice, $\mathcal{M}$ is used exactly once to generate the representation θ. Given such a representation, $\hat{f}(q;\theta)$ answers any RCQ, q, without further access to the database. For instance, in [24], $\mathcal{M}$ is a mechanism that outputs noisy counts of cells of a 2-dimensional grid overlaid on D. Then, to answer an RCQ q, $\hat{f}(q;\theta)$ takes the noisy grid, θ, and the RCQ, q, as inputs and returns an estimate of $f(q)$ using the grid. The objective is to design $\mathcal{M}$ and $\hat{f}$ such that the relative error between $\hat{f}(q;\theta)$ and $f(q)$ is minimized, that is, to minimize $E_{\theta\sim\mathcal{M}}E_{q\sim\mathcal{Q}}[\Delta(\hat{f}(q;\theta), f(q))]$, where $\Delta(\hat{y}, y)$ denotes the relative error of an estimate $\hat{y}$ when the true answer is y.

Related Work. Most related work create a DP-compliant representation of a spatial dataset by partitioning the data domain into bins, and then publish a *histogram* with the noisy count of points that fall within each bin. At query time, the noisy histogram is used to compute answers, by considering the counts in all bins that overlap the query. When a query partially overlaps with a bin, the *uniformity assumption* is used to estimate what fraction of the bin's count should be added to the answer. This is referred to *domain partitioning*, commonly adopted by existing work. For example, [24] proposes partitioning the domain into grid cells and releasing a noisy count for each cell. QuadTree [8] first generates a quadtree, and then employs the Laplace mechanism to inject noise into the point count of each node and Privtree [34] is another hierarchical method, but allows variable node depth in the indexing tree.

3.2 SNH Overview

Rather than creating a hand-designed data representation, Spatial Neural Histograms (SNH) learns neural networks to answer RCQs while preserving differential privacy, using neural network parameters as the data representation. Results from NeuroSketch, discussed in Sec. 2, show that learning can exploit data patterns to accurately and compactly represent the data. As such, learning can be used to combat data modelling errors, present in the domain partitioning methods and amplified due to the impact of differential privacy noise. Nonetheless, the challenge here, compared with NeuroSketch, is that the training process has to satisfy ε-DP. This means that the training process cannot ask an arbitrary number of queries from the database (as asking each query from the database consumes privacy budget) to train neural networks. As such, two new steps are introduced in the training

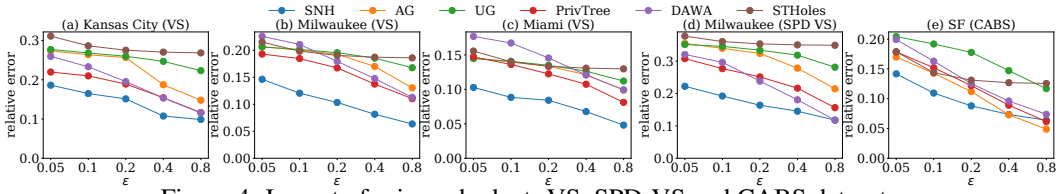

Figure 4: Impact of privacy budget: VS, SPD-VS and CABS datasets

process, specifically, data collection and data augmentation. Thus, the SNH framework, illustrated in Figure 3, consists of three steps: (1) Data collection, (2) Model Training, and (3) Model Utilization. We provide a summary of each step below.

Data Collection. This step partitions the space into non-overlapping RCQs that are directly answered with DP-added noise. The advantage of non-overlapping queries is that, due to the parallel composition property of differential privacy, one only need to spend privacy budget once to answer any number of non-overlapping queries. The output of this step is a *data collection query set*, Q_D, and a set Y_D which consists of the differentially private answers to RCQs in Q_D. This is the only step in SNH that accesses the database. In Fig. 3 for example, the query space is partitioned into four RCQs, and a differentially private answer is computed for each.

Training. Our training process consists of two stages. First, we use *spatial data augmentation* to create more training samples based on Q_D. An example is shown in Fig. 3, where an RCQ covering both the red and yellow squares is not present in the set Q_D, but it is obtained by aggregating its composing sub-queries (both in Q_D). The data augmentation is important because, due to differential privacy, one can collect a limited number of queries for training. Thus, this data augmentation step utilizes the characteristic of database queries (e.g., that two queries can be combined to create a new query) to augment the training set. Second, the augmented training set is used to train a function approximator $\hat{f}$ that captures f_D well. $\hat{f}$ consists of a set of neural networks, each trained to answer different query sizes. This, similar to NeuroSketch, follows our query specialization principle discusses in Sec. 2, where multiple models are trained, each specialized to answer specific quereis.

Model Utilization. After training, models are used to directly answer queries. For a query, we first decide which model to use to answer queries from based on the query range, and use that model to obtain the final query answer estimate.

3.3 Experimental Results

We compared SNH with various existing methods, PrivTree [34], Uniform Grid (UG) [24], Adaptive Grid (AG) [24] and Data and Workload Aware Algorithm (DAWA) [15]. all of which release two-dimensional noisy histograms but utilize various means of domain partitioning (no existing method utilizes learned data representation), as well as an additional baseline, STHoles, which is a modification of [4], a non-private workload-aware algorithm, to satisfy DP. The experiments consider answering RCQs on tree datasets: CABS [23] is derived from the GPS coordinates of approximately 250 taxis collected over 30 days in San Francisco, VS is derived from anonymized location signals of cell-phones across the US collected by Veraset [2] and SPD-VS is a processed version of VS that only keeps users stay-points following [29] (i.e., removes the points where users are driving/walking/etc.).

Fig. 4 presents the error of SNH and competitors when varying ε for test datasets VS, SPD-VS and CABS. Recall that a smaller ε means stronger privacy protection.For VS and SPD-VS, we observe that SNH outperforms the state-of-the-art by up to 50% at all privacy levels (Fig. 4 (a)-(d)). This shows that SNH is effective in utilizing machine learning to improve accuracy of privately releasing proprietary datasets. Fig. 4 (e) shows that SNH also outperforms other approaches on CABS dataset in almost all settings, the advantage of SNH being more pronounced for smaller ε values. Stricter privacy regimes are particularly important for location data, since such datasets are often released at multiple time instances with smaller privacy budget per release.

4 Answering Aggregate Queries on Incomplete Relational Databases

In this section, we provide an overview [32] which shows an application of NeuroDB for answering aggregate queries on incomplete datasets.

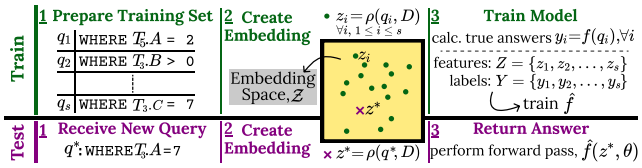

Figure 5: NeuroComplete Framework

4.1 Preliminaries

Problem Definition. The setting here is more general than the previous two scenarios discussed. Here we consider a general relational database setting. Consider a relational database, D, with k tables, T_1, ..., T_k. Foreign key relationships connect (some of) the tables. Each table has a primary key which uniquely identifies the rows within each table. We consider aggreagate queries, q, on this database. Informally, q asks for an aggregation of an attribute in some table, where the records in the table are filtered based on some predicate (optionally containing `JOIN` or `GROUP BY` clauses).

We consider the case when we only have access to a subset of records, $\bar{T}_i$ of the table T_i for some $i \in \{1, ..., k\}$. We refer to table T_i as *incomplete* or *partially observed* and refer to tables T_j, $j \neq i$ as *complete* or *fully observed*. We let the incomplete database $\bar{D}$ be the database consisting of $\bar{T}_i$ and T_j for all j. We often refer to D as the true database and $\bar{D}$ as observed database. The *observed query function*, $\bar{f}(q)$ is the function that takes a query as an input and outputs the query answers based on the observed databse. We consider the case when the observed database is a biased sample of the true database, i.e., $\mathbb{E}_{\bar{D}\sim D}[\bar{f}(q)] \neq f(q)$. Thus the error in answering queries on the observed database isn't only due to the variance in sampling, but also due to its bias. We denote by $n = |T_i|$ and $\bar{n} = |\bar{T}_i|$, the size of the observed table and true table, respectively.

The goal is to, given the observed database, $\bar{D}$, answer a query q so that its answer is similar to $f(q)$. However, performing the query on the observed database, $\bar{D}$, provides an inaccurate answer $\bar{f}(q)$. Instead, using $\bar{D}$, we train a model $\hat{f}(.;\theta)$ that takes the query as an input and outputs an estimate of its answer. The model is trained given only $\bar{D}$, but its answer is expected to be similar to performing queries on D. The asked queries can have arbitrary predicates, a fixed aggregation function `AGG` and a fixed measure attribute M. Let $\mathcal{Q}$ be the set of all such queries from a query workload. Given access only to an observed incomplete database $\bar{D}$, our goal is to train a model, $\hat{f}$, so that $\frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} |\hat{f}(q;\theta) - f(q)|$ is minimized, where f is the query function corresponding to the complete database D. We follow the setup of [12] and ask the users to (1) annotate tables with missing records and (2) annotate rows that have *complete* foreign key relationships, where for such rows, the foreign keys are not missing.

Related Work. There has been recent effort in answering queries on incomplete datasets [12, 21, 30, 28, 25, 6]. With existing approaches generating data either through data imputation [12, 30, 28, 25, 6] or synthetic data generation. The only approach directly applicable to this setting is ReStore [12] which considers synthesizing entire records to *complete* the observed dataset. ReStore [12] utilizes foreign key relationships to synthesize new data records, and the synthetically generated data is added to the database. After data generation, the query is answered as in a typical relational database.

4.2 NeuroComplete Overview

NeuroComplete proposes a fundamentally different approach to answering queries on incomplete data. Rather than generating new data it proposes to learn a model that directly estimates the query answers. This is done in three steps. First, it generates a set of training queries for which accurate answers can be computed given the incomplete dataset. Next, NeuroComplete extracts a set of features for each of these queries. Each feature corresponds to the contextual information available about the query answers in the database, and is computed based on how related a database record is to the query. Finally, NeuroComplete trains a neural network in a supervised learning fashion to learn a mapping from the embedding space (i.e., query features) to query answers. The learned model then generates accurate answers to new queries at test time, exploiting the generalizability of the learned model in the embedding space.

Figure 5 shows an overview of this approach. NeuroComplete embeds queries into a space $\mathcal{Z}$ and trains a model, $\hat{f}$, from $\mathcal{Z}$ to query answers. To do so, NeuroComplete defines an embedding function ρ that takes a query q as an input and outputs an embedding z. To answer any query, q, we first find

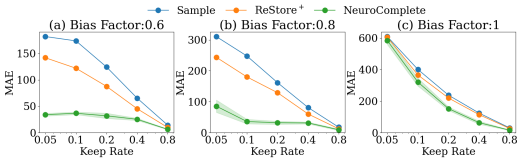

Figure 6: Results for AVG Rental Prices

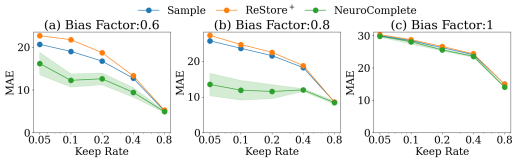

Figure 7: Results for AVG Response Rate

$z = \rho(q)$ and then provide the estimate $\hat{f}(z;\theta)$ for the query answer. The input to the neural network is a query embedding, which represents the query in terms of the observed information related to the query. Intuitively, the embedding function ρ aggregates the observed database rows based on how related they are to the query, to represent the query in terms of such relevant information. As shown in Fig. 5, during training, NeuroComplete follows three steps. (1) It creates a set, Q, of queries for the purpose of training whose answer can be accurately obtained from the observed database. Intuitively, any query that is "restricted" such that its answer only depends on the data in the incomplete database can be answered accurately. (2) It uses the embedding function, ρ, to find the query embedding for the queries in Q, and (3) uses the queries together with their answer (computed on the observed database) to train a neural network $\hat{f}$ in a supervised learning setting. The neural network learns a mapping from the embedding space to query answers. To answer a query, NeuroComplete first finds its query embedding and performs a forward pass of the trained neural network with the embedding as its input to provide an estimate of the query answer.

4.3 Experimental Resuls

We compare NeuroComplete with Restore [12], which generates new data to complete the database and answers queries based on the generated data, as well as Sample which is the method that merely answers queries on the observed dataset. The results are on a database of Airbnb listings (containing information such as the apartment type, price, its neighbourhood and landlord) obtained from [1]. We consider two queries, asking for AVG rental prices of listings and AVG response rate of landlords for different predicates (e.g., listings with different numbers of rooms). To evaluate the methods, we select a biased portion of the original dataset (keep rate denotes the percentage of the original dataset that was kept) where the bias is introduced based on rental prices and response rates (e.g., only apartments with higher rental prices are kept). Bias factor denotes how non-uniform the samples are (the larger the bias factor, the more the bias). All the algorithms only have access to this biased subset, but the goal is to predict the actual rental prices on the original dataset.

Figs. 6-7 compare NeuroComplete with other methods across settings for AVG queries. Each figure shows, for a setting, how the error changes for different keep rates and bias factors. For NeuroComplete, the shaded area shows one standard deviation above/below error, where standard deviation is over 5 training runs. We observe that NeuroComplete outperforms the baselines across settings in almost all cases, improving accuracy of state-of-the-art by up to a factor of 4. Furthermore, NeuroComplete is the most effective when bias factor is less than 1 and when keep rate is less than 80%. When bias factor is 1, NeuroComplete does not see enough variation in query answers during training to be able to accurately extrapolate to unseen queries. On the other hand, when keep rate is 80%, Sample itself is very accurate, and inherent modeling errors do not allow for much improvement for NeuroComplete over observed values.

5 Conclusion

We have presented the NeuroDB framework, and discussed its application to approximate query processing, privacy-preserving query answering and query answer on incomplete datasets, showing the benefits of this framework in the three application scenarios. The discussion has shown how different query representations and training procedures can be utilized, together with the NeuroDB framework, to answer queries in a variety of applications. Future work includes applying the furthermore to different application scenarios such as similarity search, studying how the model should be updated in the case of dynamic datasets and extending the theoretical understanding of why and in which scenarios the NeuroDB framework performs well.

Acknowledgements

This research has been funded in part by NSF grant IIS-2128661. Opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any sponsors, such as NSF.

References

[1] Housing dataset. https://public.opendatasoft.com/explore/dataset/airbnb-listings/table/?disjunctive.host_verifications&disjunctive.amenities&disjunctive.features. Accessed 7/22.

[2] Veraset website. https://www.veraset.com/about-veraset, 2020. Accessed: 2020-10-25.

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European conference on computer systems*, pages 29–42, 2013.

[4] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, page 211–222, New York, NY, USA, 2001. Association for Computing Machinery.

[5] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)*, 32(2):9–es, 2007.

[6] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *SIGMOD*, 2016.

[7] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1–3):1–294, Jan. 2012.

[8] G. Cormode, C. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012.

[9] C. Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[10] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan. Learning memory access patterns. volume 80 of *Proceedings of Machine Learning Research*, pages 1919–1928, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[11] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 171–182, 1997.

[12] B. Hilprecht and C. Binnig. Restore-neural data completion for relational databases. In *SIGMOD*, pages 710–722, 2021.

[13] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7), 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*, pages 489–504, 2018.

[15] C. Li, M. Hay, G. Miklau, and Y. Wang. A data- and workload-aware algorithm for range queries under differential privacy. *Proc. VLDB Endow.*, 7(5):341–352, Jan. 2014.

[16] X. Liang, T. Zou, B. Guo, S. Li, H. Zhang, S. Zhang, H. Huang, and S. X. Chen. Assessing beijing's pm2. 5 pollution: severity, weather impact, apec and winter heating. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2182):20150257, 2015.

[17] Q. Ma and P. Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1553–1570, 2019.

[18] M. Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

[19] R. O. Nambiar and M. Poess. The making of tpc-ds. VLDB '06, page 1049–1058. VLDB Endowment, 2006.

[20] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 985–1000, 2020.

[21] L. Orr, M. Balazinska, and D. Suciu. Sample debiasing in the themis open world database system. In *SIGMOD*, 2020.

[22] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1461–1476, 2018.

[23] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. Crawdad data set epfl/mobility (v. 2009-02-24), 2009.

[24] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 757–768. IEEE, 2013.

[25] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *VLDB*, 2017.

[26] R. R. Schmidt and C. Shahabi. Propolyne: A fast wavelet-based algorithm for progressive evaluation of polynomial range-sum queries. In *International Conference on Extending Database Technology*, pages 664–681. Springer, 2002.

[27] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing for data exploration using deep generative models. In *2020 IEEE 36th international conference on data engineering (ICDE)*, pages 1309–1320. IEEE, 2020.

[28] R. Wu, A. Zhang, I. Ilyas, and T. Rekatsinas. Attention-based learning for missing data imputation in holoclean. *MLSys*, 2020.

[29] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining individual life pattern based on location history. In *2009 tenth international conference on mobile data management: Systems, services and middleware*, pages 1–10. IEEE, 2009.

[30] J. Yoon, J. Jordon, and M. Schaar. Gain: Missing data imputation using generative adversarial nets. In *ICML*, 2018.

[31] S. Zeighami, R. Ahuja, G. Ghinita, and C. Shahabi. A neural database for differentially private spatial range queries. *Proc. VLDB Endow.*, 15(5):1066–1078, jan 2022.

[32] S. Zeighami, R. Seshadri, and C. Shahabi. A neural database for answering aggregate queries on incomplete relational data. *Transactions of Data Engineering (TKDE)*, 2023.

[33] S. Zeighami, C. Shahabi, and V. Sharan. Neurosketch: Fast and approximate evaluation of range aggregate queries with neural networks. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.

[34] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proceedings of the 2016 International Conference on Management of Data*, pages 155–170, 2016.