Uncertainty-aware Cardinality Estimation by Neural Network Gaussian Process

Kangfei Zhao
The Chinese University of Hong Kong
kfzhao@se.cuhk.edu.hk

Zongyan He
Renmin University of China
hzy9819@ruc.edu.cn

Jeffrey Xu Yu
The Chinese University of Hong Kong
yu@se.cuhk.edu.hk

Hao Zhang
The Chinese University of Hong Kong
hzhang@se.cuhk.edu.hk

Abstract
Deep Learning (DL) has achieved great success in many real applications. Despite its success, there are some main problems when deploying advanced DL models in database systems, such as hyper-parameters tuning, the risk of overfitting, and lack of prediction uncertainty. In this paper, we study cardinality estimation for SQL queries with a focus on uncertainty, which we believe is important in database systems when dealing with a large number of user queries on various applications. With uncertainty ensured, instead of trusting an estimator learned as it is, a query optimizer can explore other options when the estimator learned has a large variance, and it also becomes possible to update the estimator to improve its prediction in areas with high uncertainty. The approach we explore is different from the direction of deploying sophisticated DL models in database systems to build cardinality estimators. We employ Bayesian deep learning (BDL), which serves as a bridge between Bayesian inference and deep learning. The prediction distribution by BDL provides principled uncertainty calibration for the prediction. In addition, when the network width of a BDL model goes to infinity, the model performs equivalent to Gaussian Process (GP). This special class of BDL, known as Neural Network Gaussian Process (NNGP), inherits the advantages of Bayesian approach while keeping universal approximation of neural network, and can utilize a much larger model space to model distribution-free data as a nonparametric model. We show that our uncertainty-aware NNGP estimator achieves high accuracy, can be built very fast, and is robust to query workload shift, in our extensive performance studies by comparing with the existing approaches.

1 Introduction
The learning approaches are shifting from the traditional ML (Machine Learning) models (e.g., KDE, GBDT) to DL (Deep Learning) models (e.g., NN, MSCN, VAE, MADE, Transformer, SPN, LSTM). The shifting is motivated by the powerful approximation capability of neural networks in end-to-end applications and the high efficiency of DL frameworks. Compared with traditional ML, DL has achieved a great improvement on estimation accuracy, and more and more advanced DL architectures are devised to improve the performance of the learning tasks with deeper layers to pursue more powerful modeling capability. In database systems, the DL models have been extensively studied for query optimization [45], index recommendation [9], view materialization [41], and cardinality estimation [13, 14, 24, 31, 32, 56, 66, 67]. Despite the success of DL approaches, the complex DL approaches incur some main problems in general.

The first is hyper-parameters, on which the performance of DL models highly relies, including the training hyper-parameters and network architecture configurations. Note that onerous effort of parameter tuning must be paid to pursuing satisfying performance, which is labor-intensive. Although autoML tools [30] have been developed to avoid human-in-the-loop, deploying such tools in database systems needs great effort as searching a suitable hyper-parameter combination needs a huge number of trials with high computation cost. It is worth mentioning that a well-tuned model for one database is difficult to transfer to other databases, which means that the retraining/tuning needs repeating when the underlying data changes significantly.

The second is a high risk of overfitting that DL models are exposed to. Subject to a particular family of function designed, general DL models are indexed by a large number of parameters, fully fitted by the training data. It is assumed testing data is from the same underlying distribution of the training data, otherwise the prediction may have a large variance. Many regularization techniques can be used, but they can only alleviate the problem to some degree. Collecting more data to enhance training helps to reduce the variance. But this means a higher cost for acquiring the data/ground truth and training.

The third is prediction belief, which is an important issue we focus on in this work. The issue is that DL models cannot capture and convey their prediction belief, namely, how probably their prediction is accurate or how much is the prediction uncertainty. The uncertainty comes from different sources, for example, the noise of the training data, the dissimilarity of the test data from the training data, the mismatching of the model class to the data to be described. For classification tasks, DL models predict the probability distribution that one data point is associated with the candidate classes by a softmax function, which is shown to be over-confident on the most likely class [22]. For regression tasks (e.g., cardinality estimation), DL models can only output a scalar value without any uncertainty measurements of the prediction, such as variance and confidential interval. It is highly desirable to avoid situations where we have no choices but trust the DL predictions being made in database systems. In other words, as a database system to support a large number of users in various applications, where user queries may be different and databases will be updated from time to time, what database systems require is not only an
accurate model prediction, but also an indication of how much the predictions can be trusted regarding the learned model.

In this paper, we study cardinality estimation for SQL queries with selection, projection, and join. We focus on uncertainty. The approach we take can also address the hyper-parameter and risk of overfitting. Table 1 summarizes the recent ML/DL approaches studied for AQP (Approximate Query Processing) (the top three) and cardinality estimation (the bottom eight). These approaches are categorized into supervised learning approaches and unsupervised learning approaches. The supervised learning approaches are query-driven, which learn a function that maps the query features to its cardinality. The unsupervised learning approaches are data-driven, which learn the joint probability distribution of the underlying relational data. The supervised learning approaches are easy to deploy with relatively low training and prediction overhead, however, they lack robustness to shifting query workloads. On the contrary, the unsupervised learning approaches are robust to different query workloads. But, building a model consumes many resources as with relatively low training and prediction overhead, however, they which learn the joint probability distribution of the underlying relational data. The supervised learning approaches are easy to deploy with relatively low training and prediction overhead, however, they lack robustness to shifting query workloads. On the contrary, the unsupervised learning approaches are robust to different query workloads. But, building a model consumes many resources as

Table 1: ML/DL Approaches for AQP and Cardinality Estimation

| Approach | Supported Queries | Learning Strategy | Model | Model Update |
|----------|-------------------|------------------|-------|--------------|
| DBEst [42] | precomp. join, num., cate | supervised & unsupervised | KDE & GBDT | ✓ |
| DeepDB [25] | precomp. join, num., cate | supervised | SPN | ✓ |
| Thirumuruganathan et al. [58] | num., cate | unsupervised | VAE | ✓ |
| Kipf et al. [32] | PK/FK join, num., cate | supervised | MADE | ✓ |
| Sun et al. [56] | PK/FK join, num., cate, estr. | supervised | Tree LSTM | ✓ |
| Nara [67] | precomp. join, num., cate | unsupervised | MADE, Transformer | ✓ |
| NeuroCard [66] | Full outer join, num., cate | unsupervised | MADE, Transformer | ✓ |

Figure 1: From ML/DL estimators To NNGP (GP & BDL) a nutshell, BDL imposes a prior distribution on the neural network weights, and the derived models are ensembles of neural networks in a particular function space, which output a distribution as the prediction. The prediction distribution provides principled uncertainty calibration for the prediction. This uncertainty helps the model to improve itself explicitly, by collecting the information of testing data with higher uncertainty and retraining/fine-tuning the model. In addition, the prediction distribution by BDL is robust to the overfitting problem. The parameters and hyper-parameters of BDL models are much fewer (i.e., the statistics of the prior distribution and the prior type, respectively), in versus to the parameters and hyper-parameters that need in DL models (i.e., the weights of the neural networks and the network architecture configurations, number of layers, number of hidden units in each layer), and in DL training configurations (i.e., optimization algorithm, learning rate, batch size, epochs, etc.). When the network width of the BDL model goes to infinity, the model performs equivalent to Gaussian Process (GP) [48], named Neural Network Gaussian Process (NNGP). Exact Bayesian inference can be used to train this special GP as a lightweight cardinality estimator, while offering a more powerful generalization capability than a finite wide neural network. NNGP keeps the flexible modeling capability of deep learning, while offering robustness and interpretability by Bayesian inference. Fig. 1 delineates the extraordinary standpoint of our NNGP estimator among the existing ML/DL-based estimators in Table 1, for cardinality estimation. Here, KDE and GBDT are classical ML models while SPN is a new type of probabilistic graphical model with deeper layers. DL models are associated with different neural network architectures. A BDL model is a special kind of DL model in which the neural network parameters are probability distributions. We explore a new model, named NNGP, which is the BDL model with infinite wide hidden layers, and can be built using GP.

In this work, different from the direction of deploying sophisticated DL models in database systems, we employ Bayesian deep learning (BDL) to build cardinality estimators, where BDL serves as a bridge between Bayesian inference and deep learning [61]. In

| Approach | Supported Queries | Learning Strategy | Model | Model Update |
|----------|-------------------|------------------|-------|--------------|
| DBEst [42] | precomp. join, num., cate | supervised & unsupervised | KDE & GBDT | ✓ |
| DeepDB [25] | precomp. join, num., cate | supervised | SPN | ✓ |
| Thirumuruganathan et al. [58] | num., cate | unsupervised | VAE | ✓ |
| Kipf et al. [32] | PK/FK join, num., cate | supervised | MADE | ✓ |
| Sun et al. [56] | PK/FK join, num., cate, estr. | supervised | Tree LSTM | ✓ |
| Nara [67] | precomp. join, num., cate | unsupervised | MADE, Transformer | ✓ |
| NeuroCard [66] | Full outer join, num., cate | unsupervised | MADE, Transformer | ✓ |
Contribution. The main contributions of this paper are summarized as follows. 1 We employ an advanced BDL model, named Neural Network Gaussian Process (NNGP), to build cardinality estimators for relational database systems. Our NNGP estimator can support range queries on single relations and equi/theta-join on multiple relations for a database. To the best of our knowledge, this is the first exploration of BDL approaches for database applications. 2 As the first DL-based cardinality estimator which supports principled uncertainty estimation, we investigate the uncertainty that NNGP captures, compared with existing uncertainty quantification approaches for DL. 3 We conduct extensive experimental studies to verify the effectiveness and efficiency of NNGP estimator via comparison with recent ML/DL based estimators. The NNGP estimator distinguishes from existing estimators by its swift model construction and prediction, robustness to workload shifts, and appealing accuracy.

Roadmap. §2 gives the problem statement. In §3, we give an overview of the NNGP estimator, and introduce Bayesian learning, GP and NNGP in §4. Then, we elaborate on the uncertainty calibration of NNGP in §5. §6 reports the experimental results. Finally, we review the related works in §7 and conclude the paper in §8.

2 Problem Statement

A relational database consists of a set of relations, \( \{ R_1, R_2, \cdots, R_N \} \), where a relation \( R_i \) has \( d_i \) attributes such as \( R_i = (A_1, \cdots, A_{d_i}) \). Here, an attribute \( A_j \) is either a numerical attribute in a domain with given range \([\min_j, \max_j]\) or a categorical attribute with a discrete finite domain \( \{c_1, c_2, \cdots, c_{m_j}\} \).

We study cardinality estimation for the select-project-join SQL queries with conjunctive conditions. A selection on an attribute is either a range filter (i.e., \([lb_j, ub_j]\) denoting the condition \( lb_j \leq A_j \leq ub_j \)), if the attribute is numerical, or a \( IN \) filter (i.e., \( A_j \in C \), \( C \subseteq \{c_1, c_2, \cdots, c_{m_j}\} \)), denoting the condition \( \exists k \in C, A_j \equiv c_k \), if the attribute is categorical. A projection can be on any attributes. A join can be equi/theta-join. For joining over numerical attributes, the join condition can be \((<, \leq, =, \geq, \neq)\). For joining over categorical attributes, the join condition is either \( = \) or \( \neq \), as the categorical domain is order-free. The primary-foreign key join, or PK/FK-join, is treated as a special equi-join with the extra constraints. We also support self-joins, which are conducted by renaming the same relations. An example is given below.

\[
\sigma_{(100 \leq R_1.A_1 \leq 200) \land (R_2.A_1 \in \{15,20,25\})}(R_1 \bowtie_{R_1.A_1 \leq R_2.A_1} R_2)
\]

where \( \sigma \) is the select operator and \( \bowtie \) is a join operator. It is worth mentioning that we support general select-project-join SQL queries. All the existing learned estimators do not support joins other than PK/FK joins. For example, [66] does not support cyclic join queries and selection conditions on join attributes and [25] does not support multiple selection conditions on join attributes.

The cardinality of an SQL query \( q \) is the number of resulting tuples, denoted as \( c(q) \). To learn a cardinality estimation, like the existing work, we require a set of joinable attribute pairs \((R_1.A_1, R_2.A_2), \cdots\) in addition to a set of relations \( \{R_1, \cdots, R_N\} \). For example, two relations \( \{R_1, R_2\} \) with joinable attribute pairs \( \{R_1.A_3, R_2.A_3\} \).

The Problem Statement: Given a set of relations and a set of joinable attribute pairs, learn a model \( M : q \mapsto \mathbb{R} \) from a training query set \( \{(q_1, c(q_1)), (q_2, c(q_2)), \cdots\} \), where \( q_i \) is an SQL select-project-join query and \( c(q_i) \) is its actual cardinality, to predict the cardinality for unseen queries.

We use \( q \)-error to evaluate the accuracy of the estimated value.

\[
q\text{-error}(q) = \max \left\{ \frac{c(q) - \hat{c}(q)}{\hat{c}(q)} \right\}
\]

Intuitively, \( q \)-error quantifies the factor by which the estimated count \( \hat{c}(q) \) differs from the true count \( c(q) \). It is symmetrical and relative so that it provides the statistical stability for true counts of various magnitudes. Here, we assume \( c(q) \geq 1 \) and \( \hat{c}(q) \geq 1 \).

We learn an uncertainty-aware model, and we do not make any assumptions on data and query, e.g., the distribution of attribute values, the independence of attributes and relations, the distribution of the selection/join conditions of the queries, etc.

Query Encoding: Following the existing work, a select-project-join query we support can be encoded by a fixed length vector. The encoding consists of two parts: the selection conditions and the join conditions. The two parts are encoded separately and concatenated as follows.

\[
\begin{array}{cccccccc}
\text{R}_1 \text{ selections} & \text{R}_2 \text{ selections} & \text{joins} \\
0.25 & 0.5 & \cdots & 1 & 1 & 0 & 1 & 0 & \cdots & 1 & 1 & 0 & 0 & 0 \\
\text{R}_1.A_1 & \text{R}_2.A_2 & (\text{R}_1.A_3, \text{R}_2.A_3) \\
\end{array}
\]

For selection condition, the encoding of all the attributes in all the relations of the schema are concatenated by a fixed order (e.g., lexicographical order). In a similar manner, for the join conditions, the encoding of all the join pairs are concatenated.

The selection conditions are specified on numerical/categorical attributes. For a range filter, \( lb_j \leq A_j \leq ub_j \), on a numerical attribute \( A_j \), we normalize \( lb_j \) and \( ub_j \) to \([0, 1]\) by mapping \([lb_j, ub_j]\) to \([\min_j, \max_j, \min_j, \max_j] \), where \([\min_j, \max_j] \) is the domain of the attribute \( A_j \). Thus, the representation is the interval of two real values. For an \( IN \) filter, \( A_j \in C \), on a categorical attribute \( A_j \), where \( C \) is a subset of the attribute \( A_j \)’s domain \( \{c_1, c_2, \cdots, c_{m_j}\} \), a straightforward encoding is to build an \( m \)-dim bitmap where its \( k \)-th bit is 1 if \( c_k \in C \), otherwise 0. This binary representation is effective for attributes with small domain, however, it is difficult to scale on a large domain where the predicate vector is high-dimensional and sparse. Therefore, for a large domain, we adopt the factorized bitmap [66], i.e., slicing the whole bitmap to chunks, and converting each chunk into corresponding base-10 integer. Finally, the selection condition is represented losslessly by \( [m/s] \) integers, where \( s \) is the length of the chunk.

Regarding join conditions, for each joinable attribute pair \((A_i, A_j)\), we use a 3-bit bit-map to encode the join condition on this pair, corresponding to the 3 comparison operators, \(<, =, \geq\), respectively, where ‘1’ denotes there is a comparative condition on \((A_i, A_j)\). For example, \( A_i < A_j \), \( A_i \geq A_j \), and \( A_i \neq A_j \) are encoded as ‘001’, ‘011’ and ‘101’, respectively. The bit-map ‘000’ denotes that the query is free of join condition on the pair.

3 An Overview

We present an NNGP overview for cardinality estimation that learns a DL model using GP. We discuss NNGP from the viewpoints of
standard neural network and GP, where NNGP exhibits the approximation capability of neural network and can be solved by exact Bayesian inference as a regular GP. Such NNGP properties enable our cardinality estimator to be robust, lightweight, and uncertainty-aware, compared with the existing DL-based estimators. We show the differences between a neural network and NNGP in Fig. 2.

On the left, Fig. 2 shows a standard fully-connected neural network, which is the building block of all the DL-based estimators in Table 1. The hidden layer is a weighted linear transformation with nonlinearity that transforms input representation to output in a layer by layer fashion. Given an empirical loss function, the parameters (i.e., the weights of the linear transformation) fit to given training data (i.e., the vectorized relational data or SQL query regarding cardinality estimation) by forward-backward propagation algorithm. As a parametric model, the prediction on new input is determined by the learned parameters. Theoretically, neural network is able to approximate any given continuous function [26, 27], and achieve an arbitrary small approximation error with infinite wide layers.

On the right, Fig. 2 shows NNGP, which is a special class of Bayesian DL model, whose hidden layer has an infinite number of neurons. In statistical learning, Bayesian inference is a principled way to describe prediction belief. Bayesian DL is derived from developing Bayesian inference on modern DL [61]. Specifically, prior distributions are imposed on the parameters of the neural network. In other words, given a set of training data, the posterior distribution of parameters is inferred by Bayes rule. And the prediction of a new input is a probabilistic distribution computed by Bayesian model average that ensembles all the models in the parameters space weighted by the posterior of the parameters. NNGP, as a special type of Bayesian DL, is equivalent to GP in the sense that any finite collection of outputs is a Gaussian distribution, and its output is a summation of an infinite number of i.i.d. random variables implied by Central Limit Theorem [48]. The infinite hidden neurons are composed of the set of GP basis functions, leading to a parametric, non-stationary GP kernel. Compared with regular GP with stationary kernels, this DL-based kernel is more flexible to adapt to the underlying data by exploiting the representation learning ability of DL. Our testing in §6 shows that without DL, a simple GP estimator fails to achieve an approaching or better performance, compared with the DL cardinality estimators.

From the perspective of Bayesian DL, NNGP inherits the advantages of Bayesian approach while keeping the universal approximation of neural network. With the parameter prior, NNGP converts from neural network parameter learning to the prior’s hyperparameter learning so that it overcomes the over-parametrization of neural network. Instead of betting on one parameter configuration, NNGP ensembles an infinite number of plausible neural network models in the space of a given architecture and prior family, thereby providing a robust approximation. The predictive distribution naturally conveys the prediction uncertainty regarding the model’s posterior and indicates out-of-distribution testing points.

From the perspective of GP, NNGP is a nonparametric model with the infinite number of basis functions. Here, nonparametrics does not mean the model is parameter-free, but in the sense that the model can not be specified in a finite number of parameters. As a nonparametric model, NNGP utilizes a much larger model space to model distribution-free data. In other words, it does not assume data to be modeled is i.i.d. or generated from a specified distribution as what DL models require [37]. The property of distribution-free promotes deploying database-oriented learning tasks (e.g., cardinality estimation), since real data and queries in DBMS are too large and diverse. It is difficult to assume that they are subject to some distribution where both data and queries may evolve over time. Therefore, compared to general DL models, NNGP does not require large volume training samples to approximate an i.i.d. assumption. In addition, the learning paradigm of NNGP converts from parameter learning for DL to kernel learning for GP, which avoids the approximation inference of Bayesian DL by manipulating the infinite wide neural network implicitly. More concretely, NNGP under certain neural network configuration (e.g., ReLU nonlinearity) has an analytical kernel function, which means training and prediction of this special type of DL model can be solved by pure statistical method in closed-form (i.e., exact Bayesian inference) as for a regular GP. In our experiments of §6, we verify training an NNGP estimator on the fly only consumes several seconds, up to 1-2 orders faster than corresponding DL estimators.

We show NNGP, as a kernel method, from the point of neural network enhanced kernel function in Fig. 3. In Fig. 3, we visualize two kernel matrices of NNGP estimator by taking vectorized SQL queries as the input, which are created by 500 training queries and 500 test queries. The queries for Fig. 3(a) are on a single relation forest, with 2 ~ 10 selection conditions, while the queries for Fig. 3(b) are join queries over 0 ~ 5 relations in TPC-DS. The kernel matrices are ordered by the number of selection/join conditions for the training query and the coefficient of variation of the predictive distribution for the test query. In Fig. 3, the lighter the color, the larger the inner product of the infinite hidden representations of a train-test query pair, which indicates the larger the similarity between the pair. There are two key observations. First, the uncertainty of the prediction of a test query is highly correlated with its similarity to the training queries. The lower the uncertainty, the larger the similarity to all the 500 training queries. This observation supports our intuition. Second, a query with more join conditions tends to be more dissimilar to other queries. The NNGP kernel provides a simple yet effective mechanism to compare input similarity regarding transformed infinite feature space. The knowledge of

---

1In general, exact analytical posterior distribution is intractable, thereby approximate inference such as variational inference, Markov Chain Monte Carlo are adopted.
the training data is persisted in the kernel matrix to smooth the prediction, with similarity measures between training/testing data.

We discuss NNGP with the existing DL approaches for cardinality estimation following a recent experimental study in [62] that analyzes and compares 5 learned cardinality estimators over single relation. In [62], the authors identify a set of behavior logics, as the inductive biases the learned estimators are expected to capture, namely, consistency, monotonicity, stability, and fidelity. All the query-driven DL estimators only preserve the stability as they model the estimation as a regression task. The data-driven DL estimator *NeuroCard* supports fidelity but cannot satisfy others, whereas the data-driven DL estimator *DeepDB* satisfies all the logics since it relies on hierarchical density factorization. Our NNGP estimator is a supervised learning based estimator. Thereby NNGP only supports stability like all the other query-driven regressors. However, the NNGP estimator distinguishes from the existing learned estimators from its small tuning cost and swift model construction, which are promising for fast adaption in dynamic environments.

NNGP supports Bayesian based uncertainty quantification regarding the acquired knowledge, and kernel learning enables kernel based feature understanding and selection [6, 55].

Finally, the way we use GP is different from GP used for database configuration tuning [2, 12]. First, in [2, 12], GP is used to model the concerned performance given a set of input configurations, and Bayesian optimization is used to search for a better one among a set of configurations. In our approach, the GP regressor is the cardinality estimator. Second, the GP used in [2, 12] is equipped with stationary kernel functions. For cardinality estimation with a larger feature space (i.e., the query space and a higher requirement of accurate prediction), stationary kernel functions are not flexible in terms of adapting to the input data. We use non-stationary kernel functions that belong to a particular type of neural network.

## 4 NNGP: from NN to GP

In this section, we discuss Bayesian learning, and its difference from standard learning, introduce GP and present NNGP.

### 4.1 Bayesian Learning

In standard parametric ML/DL, a model to learn is a function of input \( x, f(x, w) \), parameterized by \( w \). Learning is to optimize a specified loss function \( \mathcal{L}(f(x, w), y) \) to fit the parameter \( w \) to training data \( (x, y) \) in a training set \( (X, Y) \). In contrast to learning by optimization, Bayesian learning is to learn by marginalization. The parameter \( w \) is assumed to be a random variable drawn from a prior distribution \( p(w) \). Given the observed training data \( (X, Y) \), the posterior distribution of \( w \) can be inferred by Bayes rule (Eq. (2)).

\[
p(w|Y, X) = \frac{p(Y|X, w)p(w)}{p(Y|X)} = \frac{p(Y|X, w)p(w)}{\int p(Y|X, w)p(w)dw} \tag{2}
\]

To infer the target value for testing data \( X^* \), the predictive probability \( p(Y^*|X^*, Y, X) \) is computed by applying the probabilistic sum and product rules, assuming the training and testing are conditional independent regarding \( w \).

\[
p(Y^*|X^*, Y, X) = \int p(Y^*|X^*, w)p(w|Y, X)dw \tag{3}
\]

The predictive distribution of Eq. (3) represents *Bayesian model average* [64]. That is, instead of relying on a single prediction of one model with a single configuration of parameters, Eq. (3) ensembles all the models with all possible configurations of the parameters \( w \), weighted by the posterior of the parameters, \( p(w|Y, X) \), from Eq. (2), by marginalization of \( w \). Therefore, the predictive distribution does not depend on any specific parameter configuration. In contrast, classical training of parametric model aims to find one configuration \( \hat{w} \) that maximizes the likelihood of the observed data or minimizes the empirical loss in equivalence. In other words, the posterior distribution \( p(w|Y, X) = 1 \) for \( w = \hat{w} \) and 0 otherwise, leading to the model inference as Eq. (4).

\[
p(Y^*|X^*, Y, X) = p(Y^*|X^*, \hat{w}), \hat{w} = \arg \max_w p(Y|X, w) \tag{4}
\]

Comparing the inference of Eq. (3) and Eq. (4), if the weights posterior \( p(w|Y, X) \) has a flat distribution and \( p(Y^*|X^*, \hat{w}) \) varies significantly where the posterior has mass, the discrepancy of prediction by Bayesian model average and classical approach tends to be large. One well-known reason is the observed data is insufficient or deviating from the features of the test data, where the observed data cannot well infer the weight posterior in principle. The Bayesian learning calibrates this kind of uncertainty, a.k.a., *epistemic uncertainty*.

**Example 4.1:** A simple example of Linear Regression (LR) and Bayesian Linear Regression (BLR) is shown in Fig. 4 to illustrate the differences between standard learning and Bayesian learning. In LR (Fig. 4(a)), a linear function \( f(x) = wx + b \) is used to fit the training data points, minimizing the square error. The parameters \( w \) and \( b \) can be solved analytically by least squares or gradient descent. In BLR (Fig. 4(b)), it does not aim to solve a deterministic value for \( w, b \), instead the posterior distributions of \( w, b \) are computed, given the training data, assuming the priors of \( w, b \) are simple Gaussian distributions with zero means. The model is no longer a linear function but a random variable determined by its parameter posterior,
which forms a function space with infinite linear functions. The color lines in Fig. 4(b) indicate some samples in the function space. The derived predictive distribution (Eq. (3)) of BLR is a Gaussian distribution, thereby the shaded area in Fig. 4(b) delineates the 95%-confidential interval. We can observe that in the range with few training data, i.e., $x \in [-1.5,-0.5]$, the predictive uncertainty tends to be large, reflect to the fanout of the sampled functions and wider confidential interval. BLR only has two hyper-parameters to be set, which are the variances $\sigma^2_w$, $\sigma^2_r$ of the parameter prior. The hyper-parameters can be tuned by Bayesian model selection [53].

### 4.2 Gaussian Process

Given a set of N training data points $X = \{x_1, \cdots, x_N\}$ and a model $f(x)$, GP is used in modeling the joint distribution of the model’s predictions $f(X) = \{f(x_1), \cdots, f(x_N)\}$. Consider a simple linear model in Eq. (5), with fixed basis function $\phi(\cdot) = \{\phi_1, \cdots, \phi_m\}$, (e.g., polynomial, radial basis function), which is a projection from original input to a feature space.

$$f(X) = \sum_{j=1}^{m} w_j \phi_j(x) = \Phi w$$  \hspace{1cm} (5)

In Eq. (5), $\Phi \in \mathbb{R}^{N \times m}$ is the design matrix where $\Phi_{nj} = \phi_j(x_n)$, i.e., the value of the $j$-th basis function for the $n$-th data point. Suppose a zero mean Gaussian prior is put on the weights $w$, where $\Phi \in \mathbb{R}^{N \times N}$ is the identity matrix.

$$w \sim \mathcal{N}(0, \sigma_w^2 I)$$  \hspace{1cm} (6)

Since $f(X)$ is a linear transformation of $w$, $f(X)$ is also a Gaussian distribution as Eq. (7), where $e \sim \mathcal{N}(0, \sigma_e^2 I)$ is an extra noise on $f(X)$.

$$f(X) \sim \mathcal{N}(0, \sigma_w^2 \Phi \Phi^T + \sigma_e^2 I)$$  \hspace{1cm} (7)

Here, Eq. (7) shows the function $f(x)$, as a random variable, is a Gaussian process by definition such that for any finite selection of points $X$, $f(X)$ is a joint Gaussian distribution [43]. The covariance matrix $K_{X,X} = \sigma_w^2 \Phi \Phi^T + \sigma_e^2 I$, a.k.a., the kernel matrix, models the similarity of data points. Specifically, the entry $K_{ik} = \sigma_w^2 \phi(x_i, x_k) + \sigma_e^2 I_{ik}$ measures the similarity of points $x_i, x_k$ under basis function $\phi(\cdot)$. A simple example is that $\phi(\cdot)$ is the identity function, i.e., $\phi(x) = x$, its model is equivalent to BLR.

To make prediction for testing data points $X^* = \{x_1^*, \cdots, x_N^*\}$, we need to compute the conditional distribution $p(f(X^*)|f(X))$ as the prediction. It is also proved to be a Gaussian distribution as Eq. (8), where $K_{X,X} \in \mathbb{R}^{N \times N}$, $K_{X,X'} \in \mathbb{R}^{N \times M}$, $K_{X',X} \in \mathbb{R}^{M \times N}$, $K_{X',X'} \in \mathbb{R}^{M \times M}$ are the $X,X$ kernel, $X,X^*$ kernel and $X^*,X^*$ kernel matrices, respectively.

$$f(X^*)|f(X) \sim \mathcal{N}(\mu, C)$$  \hspace{1cm} (8)

$$\mu = K_{X',X}^T K_{X,X}^{-1} f(X)$$  \hspace{1cm} (9)

$$C = K_{X',X'} - K_{X',X}^T K_{X,X}^{-1} K_{X,X'}.$$  \hspace{1cm} (10)

Given the ground truth of the training data $X$, denoted as $Y$, and the prediction target for $X^*$ is $Y^*$, the expectation of the prediction is $\mathbb{E}[Y^*] = K_{X,Y}^T K_{X,X}^{-1} Y$, derived from Eq. (9). Assume a matrix of functions $h(X^*) = K_{X,Y}^T K_{X,X}^{-1}$, we have $\mathbb{E}[f(X^*)] = h(X^*)Y$, indicating GP regression is a weighted linear smoother over the observed target value $Y$. The weight function $h$ is determined by the train-train and train-test kernels. Meanwhile, the diagonal element of matrix $C$ in Eq. (9) measures the variance of the prediction. With this predictive Gaussian distribution, we can easily compute the $\delta$-confidential interval of $f(x^*)$ as $[\mu_{x^*} - q_{\delta} diag(C)_{x^*}, \mu_{x^*} + q_{\delta} diag(C)_{x^*}]$, where $q_{\delta}$ is the $\delta$-quantile of $\mathcal{N}(0,1)$. Intuitively, the expectation $\mathbb{E}[y^*]$ should be treated as the explicit prediction $\hat{y}$. In practical applications, as it aims to minimize the predictive loss given an empirical loss function, the prediction is to find $\hat{y}$ that minimize the expected loss by averaging the empirical loss $L_{\text{exp}}$ w.r.t. the predictive distribution as Eq. (11).

$$L_{\text{exp}} = \int L_{\text{emp}}(y^*, \hat{y}) p(y^*|x^*, Y, X) dy^*$$  \hspace{1cm} (11)

### 4.3 Neural Network Gaussian Process

GP is a stochastic process with a fixed basis function. If the basis is fixed, the model is linear w.r.t. the parameters, and the kernel function $K$ as well as the predictive distribution (Eq. (8)) are analytically tractable. The limitation of a fixed basis function is its incapability of adapting to the training data. In general, a model with an adaptive basis function (e.g., neural networks) can be a potential extension, but it is much difficult to treat it analytically [53]. To address it, the authors in [48] show that there is a special case where the neural network has an infinite number of hidden units. With such findings, some complex neural network architectures with infinite wide hidden layers are proved to be GP such as convolutional neural network [18, 49], recurrent neural network [65], attention [28] and graph neural network [29].

We discuss the foundation, the infinite wide multilayer perceptions which we use for cardinality estimation. We explain it using a single hidden layer feed forward neural network, $f(x)$, which takes $d$-dim vector $x = [x_j]^d$ as input, and predicts a scalar value $y$. Here, $\sigma(\cdot)$ is the nonlinear activation function, $b^0 = [b^0_j]^m$, $b \in \mathbb{R}$ is the bias term, and both $w^0 = [w^0_{ij}]^{m \times d}$ and $w = [w_{ij}]^{m}$ are the weights of the hidden layer and the output layer, respectively. Eq. (12)-(13) show the computation on each neuron of the hidden and output layers, where $h_j$ in Eq. (12) is the post-activation of the $j$-th hidden unit.

$$h_j(x) = \sigma(\sum_{i=1}^{d} w^0_{ij} x_j + b^0_j)$$  \hspace{1cm} (12)

$$f(x) = \sum_{i=1}^{m} w_i h_i(x) + b$$  \hspace{1cm} (13)

In classical machine learning, it optimizes the parameters $w$ and $w^0$ directly, under a specified loss function as an objective, by back propagation algorithm. Note that, even though the neural network is nonlinear, it can be regarded as a linear combination of a collection of parametric basis functions $[h_1(x) \cdots h_m(x)]$ in Eq. (13) [3]. The basis functions are parametrized by the weight $w^0$ and will be trained to adapt to the training data. Under the assumption that, for each layer, the weight and bias element parameters have i.i.d. prior densities, we have

$$w^0_{ij} \sim \mathcal{D}(0, \sigma^2_w/d), b^0_j \sim \mathcal{D}(0, \sigma^2_b)$$  \hspace{1cm} (14)

$$w_i \sim \mathcal{D}(0, \sigma^2_w/m), b \sim \mathcal{D}(0, \sigma^2_b)$$
where the prior distribution $\mathcal{D}$ can be non-Gaussian. Because the weight and bias parameters are subject to be i.i.d., and have zero mean, the hidden units $h_l(x)$ are i.i.d. bounded random variables. Following the Central Limit Theorem that, for $m$ i.i.d. random variables with bounded mean and variance, the summation of them is a Gaussian distribution when $m \to \infty$. Thus, we have $f(x)$ as an approximate Gaussian distribution when the width of output layer $m$ is large, as given in Eq. (15).

\[
f(x) \sim \mathcal{N}(0, \sigma_b^2 + \sigma_w^2 \mathbb{E}[h_1(x)^2]) \tag{15}
\]

Likewise, following the multi-dimensional Central Limit Theorem, any finite collection $f(X) = \{f(x_1), \cdots, f(x_N)\}$ have a joint multivariate Gaussian distribution, which is exactly a GP.

\[
f(X) \sim \mathcal{N}(0, K) \tag{16}
\]

\[
K = \sigma_w^2 \mathbb{E}[\Phi\Phi^T] + \sigma_b^2 I \tag{17}
\]

This GP is the Neural Network Gaussian Process (NNGP). This reveals priors over infinite wide neural network leads to an equivalence to GP. In the kernel function $K$ of Eq. (17), $\Phi$ is the design matrix given the parametric basis function $[h_1(x), \cdots, h_m(x)]$. The difference between the NNGP kernel (Eq. (17)) and the standard liner model’s kernel (Eq. (7)) is that the NNGP kernel needs to compute the expectation of the product of the design matrix w.r.t. the prior distribution of the parameters, which is used to define the basis function. This enables NNGP to be not only a model ensemble in the space of the linear parameters $\omega$ but also in the space of the basis function parameters $\omega^0$. By recursively applying Central Limit Theorem, the kernel of deep neural network is induced [38] in Eq. (18), where $f_j^l$ is the pre-activation of the $j$-th hidden unit in the $l$-th layer and $\Phi^l$ is the design matrix defined by the 0-th to $l$-th layers of the neural network. The base case kernel $K^0$ in Eq. (19) is equivalent to the kernel of BLR.

\[
K^l = \mathbb{E}[f_j^l(X)f_j^l(X)] = \sigma_w^2 \mathbb{E}[f_j^{l-1}(X)\Phi^{l-1}\Phi^l] + \sigma_b^2 I \tag{18}
\]

\[
K^0 = \sigma_w^2 XX^T + \sigma_b^2 I \tag{19}
\]

The NNGP kernel can be computed analytically under certain activation functions $\sigma(\cdot)$ (e.g., the rectified linear function ReLU [8], the error function $\text{ErF}$ [63]). To conduct inference, as the model is a standard GP, we can get the exact solution by Eq. (8).

We discuss the complexity of NNGP. As a standard GP, exact prediction needs to compute the inverse of the kernel matrix in $O(N^3)$, where $N$ is the number of training data points. Note that the kernel matrix is in closed-form and the inversion only needs to compute once in advance. For a new test data point, inference takes vector-matrix multiplication in $O(N^2)$. Quadratic time complexity w.r.t. training data is an obstacle to deploying GP model. But, with the acceleration of parallel and GPU architecture, the training and inference time is reasonable, as confirmed in our extensive experimental studies. Note that GP is usually data-efficient.

**Model design for cardinality estimation:** To test query set $Q$, we use the mean-squared-error (MSE) as the empirical loss function of NNGP in Eq. (11), which is shown in Eq. (20)-(21).

\[
\mathcal{L}_{\text{emp}}(Q) = \frac{1}{|Q|} \sum_{q \in Q} \mathcal{L}_{\text{emp}}(c(q), \tilde{c}(q)) \tag{20}
\]

\[
\mathcal{L}_{\text{emp}}(c(q), \tilde{c}(q)) = |\log c(q) - \log \tilde{c}(q)|^2 = \log^2 \frac{c(q)}{\tilde{c}(q)} \tag{21}
\]

To achieve an average low relative error, the target $c(q)$ and prediction $\tilde{c}(q)$ are transformed to logarithmic scale. It is worth noting that minimizing the average of $\log^2 \frac{c(q)}{\tilde{c}(q)}$ (Eq. (20)) is equivalent to minimize the geometric mean of q-error, and minimizing the squared-error (Eq. (21)) further imposes higher weights on larger q-error over the average due to the square [14]. When the empirical loss is squared loss, the prediction that minimizes the expected loss of Eq. (11) is the mean of the predictive distribution, i.e., Eq. (9).

**5 Calibration of Uncertainty**

In this section, we investigate the uncertainty calibration for cardinality estimation by NNGP in comparison with two existing DL approaches for uncertainty calibration that are applicable for regression tasks, namely, Deep Ensemble and Bayesian Neural Network. We introduce them below.

**Deep Ensemble** is a uniformly-weighted mixture model where each mixture is a deep neural network [36]. Each neural network treats one data point as a sample from a Gaussian distribution, in predicting the mean and variance via the final layer of the neural network. Training one neural network $w$ is to minimize its negative Gaussian log-likelihood in Eq. (22), where $\mu_w(x)$ and $\sigma_w^2(x)$ are its predictive mean and variance, respectively.

\[
-\log p_w(y|x) = \frac{\log \sigma_w^2(x)}{2} + \frac{(y - \mu_w(x))^2}{2\sigma_w^2(x)} \tag{22}
\]

The ensemble prediction is approximated as a Gaussian distribution whose mean and variance are respectively the mean and variance of $M$ neural networks. The idea of Deep Ensemble is simple whereas it needs to maintain $M$ neural networks explicitly.

**Bayesian Neural Network (BNN),** as the BDL model, quantifies the uncertainty for neural network by defining a prior belief $p(w)$ on its parameterization. The prediction is a distribution marginalizing over the posterior distribution $p(w|Y, X)$ as shown in Eq. (3). The computation of this marginalization is approximated by variational inference where $q(w)$ is a tractable variational distribution. Gal et al. in [16, 17] propose an efficient inference that relates a Bernoulli variational distribution to BNN via dropout training of the neural network. In [16, 17], inference is done by training with a dropout before weight layers and by performing dropout at test time, and the output distribution is approximated by $T$ Monte Carlo forward passes with stochastic parameter $w_t$ (Eq. (23)).

\[
p(y^*|x^*, Y, X) \approx \int p(y^*|x^*, w)q(w)dw \approx \frac{1}{T} \sum_{t=1}^{T} p(y^*|x^*, w_t) \tag{23}
\]

We have implemented above two techniques for the lightweight neural network estimator [14], as DeepEns and BNN-MCD respectively. The estimator is a two-layer multilayer perceptron with 512 hidden units. The DeepEns ensembles 5 estimators and BNN-MCD
6 Experimental Studies

In this section, we give the test setting (§6.1), and report the extensive experiments in the following facets: ① Compare the accuracy of NNGP estimator with state-of-the-art ML/DL estimators (§6.2). ② Test the training and prediction efficiency of NNGP estimator and verify it is a lightweight DL-based estimator (§6.3). ③ Validate the robustness of NNGP estimation regarding small number of training queries and unbalanced workloads (§6.4) ④ Study the application of the uncertainty that NNGP provides in active learning, which all the DL-based estimators do not support (§6.5).

6.1 Experimental Setup

Baseline Approaches. To comprehensively evaluate the effectiveness and efficiency of NNGP estimator, we compare it with 8 estimators including 5 query-driven learned estimators, 2 data-driven learned estimators and 1 traditional estimator as follows. ① Neural Network (NN) [14] is the standard fully-connected neural network with ReLU activation. ② Gradient Boosting Decision Tree (GBDT) [13, 14] is the ensembling decision regression trees by gradient boosting. ③ Multi-set Convolutional Neural Network (MSCN) [32] firstly embeds the table set, join set and predicate set by 3 separate multilayer perceptrons as the set convolutions, respectively, and then concatenate the 3 embeddings to a long vector and feed it into a final output network. ④ Tree LSTM (TLSTM) [56] is a high-end DL model originally designed for cost and cardinality estimation, by taking tree-structured query plans as its input. It is composed of three stacked layers, the embedding layer, representation layer and estimation layer. The embedding layer embeds operations (join and table scan), predicates, metadata of leaf nodes of the plan tree into vector representations. The embedding layer aggregates the representation of each node on the tree in a bottom-up fashion by Tree-Structured LSTM [57]. The estimation layer is a fully-connected neural network with sigmoid activations that finally outputs the predictions. A detailed formulation of the model can be found in [56]. ⑤ DeepDB [25] (DeepDB) is an unsupervised learning, data-driven estimator that uses Relational Sum-Product Network (RSPN) [46] to model the joint distribution of a relation. To support join queries, an ensemble of RSPNs or a joint RSPN is built, and the choice is based on independent test on the relations. SQL queries are compiled into probabilistic queries on the RSPN. ⑥ Nbaru [67]/NeuroCard [66] (NeuroCard) factorizes the joint distribution into conditional distributions and adopts deep autoregressive models such as MADE [19] or Transformer [60] to approximate the joint distribution. NeuroCard further extends Nbaru to support full outer joins. To predict the cardinalities, progressive sampling is conducted over the density model. ⑦ PostgreSQL estimator (PostgreSQL) is a build-in statistical estimator. Estimated cardinality is obtained from the EXPLAIN command of PostgreSQL. ⑧ Gaussian Process with radial basis kernel function (GP-RBF) is compared as a typical GP baseline. All the above learned estimators only support PK/FK joins.

Implementation and Settings. The NNGP estimator is built on Neural Tangents [50], which is based on Google JAX [4]. The empirical zero-mean Gaussian prior is imposed on the weights of neural networks and ReLU is used as the nonlinear activation, leading to a closed-form kernel function for Bayesian inference. There is no extra cost paid for NNGP hyper-parameters tuning. GBDT estimator is implemented by XGBoost [7] and a tree ensemble contains 32 trees. GP-RBF is implemented by the exact GP regressor of scikit-learn. For NeuroCard, we follow all the hyper-parameter configurations in its implementation based on MADE as the deep autoregressive model. NN, MSCN, and TLSTM estimators are implemented by PyTorch 1.6 [1]. We use the Adam optimizer with a decaying learning rate to train these models. For different datasets, the main hyper-parameters for training are tuned in their empirical range: learning rate ∈ [10^{-3}, 10^{-4}], epochs ∈ [50, 80, 100], mini-batch size ∈ (16, 32, 64), L2 penalty of Adam ∈ [10^{-3}, 10^{-5}]. For NN,
GBDT, GP-RBF and NNGP, their encodings of input are same, as the introduction in §2. Other models have their own input encoding due to their specific model design. Particularly, as TLSTM takes a tree-structured plan as input, we generate a left-deep tree for each join query following a total order of the relations. Since we only perform one task of cardinality estimation, one-hot physical operator encoding is simplified to the corresponding one-hot logical operator encoding. Both training and prediction are performed on a Linux server with 32 Intel(R) Xeon(R) Silver 4215 CPUs and 128G RAM.

**Datasets.** We use 4 relational datasets, 2 for single relation range queries, 2 for multi-relation join queries. forest [11] originally contains 54 attributes of forest cover type. Following [14, 21], we use the first 10 numerical attributes. The relation has about 581K of rows. higgs [11] is a physical dataset contains 7 high-level kinematic attributes of particles, collected by detectors and post-processed by scientific functions. The relation has 11M rows. TPC-H (1 GB). We use the relations supplier, orders, part and lineitem. There are 3 PK/FK join conditions. TPC-DS (2 GB). We use the relations store, item, customer and promotion, store-sales, where store-sales is the factual relation and others are the dimensional relations. There are 5 PK/FK join conditions and the schema has a cycle.

**Queries.** We construct large query workloads in the following way. For single relation forest and higgs, we generate query sets with the number of select conditions varying from 2 to D where D is the number of attributes, and generate 2,000 queries for each subset. To generate a query of d selection conditions, first we uniformly sample d attributes from all the D attributes, then uniformly sample each attribute by the data-centric distribution following [14]. For join queries, we test and report the existing baselines supported query type, i.e., PK/FK join without selection conditions on the join attributes. We generate query sets with the number of PK/FK joins, t, varying from 0 to \(|T| - 1\), where \(|T|\) is the number of involved relations. To generate a query of \(t (t > 0)\) joins, firstly a starting relation is uniformly sampled, then the query is constructed by traversing from the starting relation over the join graph in \(t\) steps. Here, for each relation of the sampled join query, additional selection conditions are drawn independently. For each \(t\), 4,000 and 3,000 queries are generated for TPC-H and TPC-DS, respectively. We only preserve unique queries with nonzero cardinality. Table 2 summarizes the 4 corresponding query sets.

### Table 2: Query Sets

| Type    | Dataset | # of Queries | # of Select/Join Cond. | Range of \(c(q)\) |
|---------|---------|--------------|------------------------|-------------------|
| Single Rel. | forest | 18,000       | [2, ⋯, 10]             | \([10^7, 10^9]\)   |
| Single Rel. | higgs  | 12,000       | [2, ⋯, 7]              | \([10^4, 10^7]\)   |
| Join     | TPC-H   | 16,000       | [0, 1, 2, 3]           | \([10^2, 10^5]\)   |
| Join     | TPC-DS  | 15,000       | [0, 1, 2, 3, 4]        | \([10^4, 10^7]\)   |

6.2 Accuracy

We investigate the estimation accuracy of NNGP estimator. For the query-centric approaches, i.e., NNGP, NN, GBDT, MSCN, TLSTM and GP-RBF, we split the queries into 60% for training, 20% for validation and 20% for testing. The split is conducted by stratified sampling on the subsets with different numbers of selection/join conditions.

Fig. 6 presents the statistical distribution of q-error of NNGP compared with the 8 baselines. In general, NNGP, MSCN and DeepDB are the top three best performed estimators. There is no an overall best estimator that consistently outperforms others under all the test circumstances. Due to the infinite wide hidden layer, the performance of NNGP consistently surpasses the finite wide NN. The performance advantage of NNGP is mainly reflected in queries of single relation (Fig. 6(a), 6(b)), and the 25%-75% quantile of q-error is within 1.5. For the join queries (Fig. 6(c), 6(d)), we speculate the join encoding and the multilayer perceptrons based kernel of q-error are simple to model the complex join queries. In contrast, the performance advantage of MSCN is reflected in the join queries rather than queries on single relation. Due to satisfaction the behavior logics, DeepDB achieves a promising 25%-75% quantile of q-error. However, we observe that there are many over/underestimated queries for forest (Fig. 6(a)) and TPC-DS (Fig. 6(d)), as the model would factorize intertwined attributes under the conditional independent assumption.

We discuss other baseline approaches. Although TLSTM is a high-end DL estimator with complex layers, its performance is not promising for a sole cardinality estimation task. We speculate TLSTM is mainly applicable for real cost estimation tasks when more features about the physical operators and meta-data of the
We compare the prediction time of\textit{NNGP} estimators with the baseline approaches. Fig. 7 presents the average elapsed time of the whole test query set on our 32-core CPU. In general, the lightweight \textit{GBDT} estimator achieves the fastest prediction and our \textit{NNGP} estimator is the second best. \

\textit{XGBoost} is a well-optimized learning system for \textit{GBDT}. The inference of \textit{NNGP} is a linear combination of the ground-truth cardinalities, where the weights are the closed-form kernels of Eq. (18). The prediction of \textit{DeepDB} is 2-3 orders of magnitude slower than that of supervised learning approaches. \textit{DeepDB} needs to perform a bottom-up pass on the tree-structured deep sum-product network. Estimation on \textit{NeuroCard} is quite slow as one query needs thousands of sampling on the model to perform a Monte Carlo integration. We noticed that in its original paper [66], the GPU executed prediction is up to 2 orders of magnitude faster than our CPU execution. That means GPU acceleration is necessary to employ \textit{NeuroCard} as an estimator available for DBMS. However, \textit{NeuroCard} also consumes large GPU memory. Training on forest is out of memory on a 16GB Nvidia V100. In terms of easy use for DBMS, lightweight estimators like our \textit{NNGP} have a great advantage. For \textit{NNGP}, \textit{NN} and \textit{GBDT}, the prediction time for queries with different number of selection/join conditions are equal, as the input vectors are of equal length. For other estimators, the more complex the query, the longer the prediction time. The prediction time of \textit{DeepDB} and \textit{NeuroCard} is also related to the database schema. In general, complex schema incurs a larger model and longer prediction time.

\textbf{Scalability of Training}. We test the scalability of \textit{NNGP} estimator in the facets of training time and memory usage, comparing with the top-2 lightweight models \textit{NN} and \textit{GBDT}. As the number of queries increases linearly, Fig. 8(a), 8(c), 8(e) and 8(g) show the training time on CPUs, where training an \textit{NN} takes tens to hundreds seconds while training \textit{GBDT} or \textit{NNGP} only takes less than 3 seconds. Although the training complexity of \textit{NNGP} is \(O(N^3)\), the exact Bayesian inference is boosted by highly paralleled basic linear algebra operations. The time complexity of \textit{NN} is \(O(TNdh)\), where \(T\), \(d\), and \(h\) are the number of iterations/epochs, dimension of input and the number of neurons in the hidden layer. Extra overhead is involved in the forward/backward propagation. Fig. 8(b), 8(d), 8(f) and 8(h) show the peak memory usage monitored in the training phase. Compared to \textit{NN} and \textit{GBDT}, \textit{NNGP} needs more memory to persist the kernel matrix, which is quadratic to the number of training queries. Fortunately, as we will show in §6.4, as a nonparametric model, \textit{NNGP} already achieves satisfying and robust performance under a small volume of training data. It is worth noting that all the other estimators are more time and memory consuming than the two-layer \textit{NN}. Given the same epoch, training of \textit{MSCN} is roughly constant time slower than that of \textit{NN} and training of \textit{TLSTM} is even much slower than that of \textit{MSCN} and \textit{NN}. For the data-driven estimators, \textit{DeepDB} and \textit{NeuroCard}, the resources consumed are directly determined by the scale of the input relations. Among all the learned estimators, the most consuming is \textit{NeuroCard}, where training on the large dataset higgs is failed within 72 hours.

\textbf{6.4 Robustness to Workload Shifts}

To study the robustness of \textit{NNGP} estimator, we train different models independently over various training workloads of forest and TPC-H, and test the models on split fixed test query sets, i.e., 20% of the
whole query sets. The test queries are evenly distributed on the number of selection conditions for forest or the number of join conditions for TPC-H. To generate various training workloads, first, we control the total number of training queries to 1,000, 2,000, 4,000, 8,000, respectively, where the queries with different numbers of selection/join conditions are evenly distributed, and the result is shown in Fig. 9(a) and 9(c). Second, the total number of training queries is fixed to 40% of the whole queries, and the fraction of queries with less/more selection or join conditions is set within \{2:8, 4:6, 6:4, 8:2\}. forest queries with 2-5 selection conditions are regarded as less conditions queries while others are more conditions. TPC-H queries with 0-1 joins are regarded as less conditions queries while others are more conditions. The testing results on the fixed 20% test queries are shown in Fig. 9(b) and 9(d).

In general, the key observation is that the NNGP estimator performs supremely robust on various workloads, regardless of the number of training queries and the fraction of different queries. In Fig. 9(a) and 9(c), even though there are only 1,000 training queries, the q-error statistics of NNGP varies within one order of magnitude compared with the model with 8,000 training queries, and even better than NN with 8,000 training queries. In contrast, due to overfitting, the q-error of its counterpart NN is degenerated drastically in the scenario of fewer training queries, which is up to 10^10. As the fraction of queries with less/more selection or join conditions shifts in Fig. 9(b) or Fig. 9(d), respectively, the performance of NNGP also oscillates slightly. As a nonparametric model, GBDT is relatively more robust than NN but has a larger q-error. We notice that GBDT underfits on forest, whose prediction has a large bias on the test queries and even on the training queries. In the experiments, we also observe that the performance of NN estimator is not stable during multiple runs, which is influenced by parameter initialization and stochastic optimization.

6.5 Uncertainty for Active Learning

Finally, we investigate leveraging the predictive uncertainty to improve the model explicitly by active learning. The key step of active learning is to select a set of informative test data to enrich the original training data and update the model [54]. The predictive uncertainty, as the model’s belief on its prediction, is an efficient and effective selection criterion. The corresponding active learning algorithm, a.k.a., uncertainty sampling [39], is to sample from the region of the data which has the most uncertainty regarding the current model, request the ground truth for these test data and retrain/update the model by original and added data. This process is repeated for several iterations under a specified stop condition, e.g., a given iteration number or sampling budget. As the existing DL-based estimators do not deliver the predictive uncertainty, we used the two uncertainty-aware DL-based estimators introduced in §5, DeepEns and BNN-MCD, as the baselines.

To conduct active learning, the entire queries are split into 40% for training an original base estimator, 20% for testing and 40% as a selection pool. The base DeepEns (5 NN estimators for the ensemble) and BNN-MCD are trained by 50 epochs. We apply 3 iterations of uncertainty sampling where each iteration 1,000 queries are drawn without replacement regarding the coefficient of variation. DeepEns and BNN-MCD are updated on their base model, and NNGP is retrained from scratch. Table 3 presents the mean-squared-error (MSE) (Eq. (21)) for the base model and those after the 3 iterations, on the test query set. NNGP and the two DL baselines are able to improve themselves introspectively via uncertainty-based active learning. Among them NNGP achieves the lowest test MSE. The MSE of NNGP and DeepEns are consistently reduced in the 3 iterations.
We briefly review ML/DL approaches in Table 1. It is complied to a product of expectations or probability queries on kernel density estimation (KDE) models and tree-based regressors ML/DL for Cardinality Estimation and AQP. ML/DL models different from Bayesian neural network. The former is a probabilistic graphical model based on the conditional independence assumption whereas the latter is the Bayesian DL model.

ML/DL for Cardinality Estimation and AQP. ML/DL models are exploited to support multiple database applications. Various types of ML/DL models serve as a cost estimator of algorithms or query plans, which are to support applications of data partitioning [15], index recommendation [9] and concurrency control [70]. [10, 33, 44, 47] propose learned index structures, which learn a cumulative distribution function of the underlying data. [34, 44, 45] design end-to-end learning-based query optimizers where [34, 45] optimize the binary join order and [44] generates the physical plans. Their approaches reformulate the dynamical programming problem of query optimization to Markov Decision Process and adopt different reinforcement learning (RL) algorithms to learn neural network models as the optimizer. [2, 12, 35, 68] adopt ML/DL to tune the database configurations, where GP-based Bayesian optimization and RL algorithms are used respectively to conduct an online tuning in [2, 12] and [35, 68]. It is worth mentioning that this paper is the first exploration of Bayesian DL in the database area.

8 Conclusion
In this paper, we explore a new simple yet effective NNGP estimator to estimate cardinality of SQL queries. We compare it with 7 baseline estimators over 4 relational datasets. In terms of accuracy, NNGP is one of the top-3 estimators, and performs best in many cases. In terms of efficiency of training, NNGP is 1-2 orders faster than the baseline estimators except GBDT. GBDT is marginally more efficient than NNGP but has a low prediction accuracy. In terms of uncertainty, NNGP can consistently improve its accuracy by uncertainty sampling via active learning, whereas BNN cannot do so. And it achieves a much smaller prediction error comparing to Deep Ensemble and BNN. In addition, NNGP is supremely robust on various workloads, and can be learned with much fewer training queries. Our source code is public available in https://github.com/Kangfei/NNGP.

Acknowledgement
We thank Zongheng Yang, the author of NeuroCard [66] for his help in testing the estimator.
[68] 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 Proc. SIGMOD’19, pages 415–432, 2019.

[69] Z. Zhao, R. Christensen, F. Li, X. Hu, and K. Yi. Random sampling over joins revisited. In Proc. SIGMOD’18, pages 1525–1539, 2018.

[70] X. Zhou, J. Sun, G. Li, and J. Feng. Query performance prediction for concurrent queries using graph embedding. Proc. VLDB Endow., 13(9):1416–1428, 2020.