Graph Based Database Generation from Query Constraints

Chun Wang
University of Michigan - Ann Arbor
Ann Arbor, Michigan
chunwc@umich.edu

Wanning He
University of Michigan - Ann Arbor
Ann Arbor, Michigan
hwanning@umich.edu

Changwen Xu
University of Michigan - Ann Arbor
Ann Arbor, Michigan
changwex@umich.edu

1 PROBLEM DESCRIPTION

Our group is working on solving constraint satisfaction problems (CSP) to generate a database table. Consider a single table with multiple rows and columns, where the actual data distribution is unknown. Our goal is to generate a synthetic table that closely resembles the distribution of the original table by utilizing queries and their corresponding cardinality results (i.e., the number of tuples that satisfy each query).

Find our project repo on: chunwangpro/CSP_Graph

1.1 Motivation

The task of database generation from queries is motivated from real-world database testing. Selecting an appropriate database product is a critical decision for commercial companies. However, due to the confidential nature of their data, companies often face limitations in using their datasets for public testing. A synthetic database, generated based on historical query sets and results, could serve as a practical solution. If this synthetic database produces query results similar to those of the original confidential data, companies could test the performance of various database products on the simulated data without risking privacy breaches. This approach enables organizations to identify the database product that best meets their requirements while safeguarding sensitive information.

1.2 Inputs/Outputs

The system takes queries' column index (e.g., 0, 1, ...), operators ($<, >, \leq, \geq, =$), values as inputs, where the cardinality is the label during training. The mathematical expression for a query can be represented as shown in Equation 1.

$$\begin{aligned} |\sigma_{col_0} \leq 3| &= 12 \\ |\sigma_{col_0} \leq 3 \ \& \ \sigma_{col_1} \leq 5| &= 10 \end{aligned} \tag{1}$$

This information can be stored more efficiently in a computer in the format shown in Table 1. For example, the first query implies that there are 12 tuples in which the values in column 0 are less than or equal to 3.

Table 1: Query Constraints Format

index	cols	ops	vals	cards
0	0	$\leq$	3	12
1	0,1	$\leq, \leq$	3, 5	10

The output of the system is a synthetic database, which is generated by our learned model, that aims to satisfy those query constraints and has silimar distribution as the ground truth.

1.3 Challenges and Contributions

Generally, a query may randomly sample k columns in a table with n columns, where $k = 1, 2$ and $n \geq 2$. Thus, each query represents a marginal distribution containing partial information about the table. The key challenge lies in utilizing these marginal distributions to infer the joint distribution. As the number of queries increases (e.g., 10^5), the computational complexity increases exponentially, requiring us to learn the data distribution as efficiently and accurately as possible under limited time and memory constraints.

Our contributions are mainly focused on the following aspects:

- Proposed a GCN-based model to learn CDF;
- Achieved comparable performance to baselines on small-scale queries;
- Successfully scaled to large query sets while maintaining accuracy;
- Demonstrated controlled computational overhead.

1.4 Task Allocation

Our work is divided into several parts. Chun, as the team leader, proposed the overall model idea and was mainly responsible for the data preprocessing. Wanning was in charge of two baselines of PGM and SMT. Changwen was responsible for the training and hyperparameter optimization of the GCN model.

2 REFERENCE/RELATED WORK

2.1 Previous Researches

There are several mainstream methods for data generation.

The symbolic CSP-based approach [2, 7] translates Approximate Query Processing (AQP) inputs into constraints with symbolic variables. However, this method does not always guarantee the generation of a unique database instance.

Conversely, the ILP-based approach [1, 5] formulates the cardinality constraints as linear equations and employs standard integer linear solvers to resolve them. While this approach provides exact solutions, it incurs a significant time overhead, with resource demands growing exponentially as the number of attributes increases.

Recent deep learning-based work [11, 13] utilizes a supervised deep AutoRegressive model to obtain the probability density function (PDF). Nonetheless, these methods are primarily effective for categorical data, exhibiting significant performance degradation when applied to numerical data.

The methods of AQP and AR models are difficult to reproduce in the short term without their open source code. So we implement two ILP-based baselines using typical constraint satisfaction solver (SMT) and least square optimizer (PGM), both of which will be introduced in later part.

2.2 New Model and New Approach in Our Work

In contrast to prior works, we not only propose a novel graph convolutional network (GCN)-based architecture but also introduce a fundamentally different learning paradigm. While previous approaches focused on learning the probability density function (PDF) through queries, we present a cumulative distribution function (CDF)-based methodology. We demonstrate that this CDF-based approach is better suited for large-scale numerical data applications, offering more controllable computational complexity.

3 METHODOLOGY

3.1 Datasets

Wine dataset [3], which includes 6497 tuples and 13 feature columns (containing all numeric types). For better visualization purpose, we randomly select three float type features: fixed acidity (3.8 to 15.9), volatile acidity (0.08 to 1.58) and citric acid (0 to 1.66).

Census dataset [8], which includes 48,000 tuples and 15 feature columns (containing numeric, string and categorical types). We apply the same pipeline here by picking two integer-type features: age (ranging from 17 to 90) and fnlwgt (final weight, ranging from 12,000 to 1.49 million), and one categorical feature: education level (contains 16 unique classes).

The two selected datasets encompass diverse data types, including integers, floating-point numbers, and categorical variables. Specifically, we utilize the large-scale Census dataset and the medium-sized Wine dataset. The heterogeneous nature and varying scales of these datasets ensure comprehensive evaluation of our model's performance and enhance the generalizability of our experimental findings.

Given the transparent nature of the dataset and our model's query-based input mechanism, we also need to generate appropriate query sets. We randomly sampling values from the dataset and applying the $\leq$ operator across columns to construct queries and obtain their corresponding cardinalities. The query set size ranges from 1 to 10^5. This range was specifically chosen for comparative analysis, as our baseline models are limited to small-scale samples, while our proposed approach demonstrates scalability to larger query sets.

3.2 Data Preprocessing

In the preprocessing phase, we focus on handling the query set. Initially, we normalize query cardinalities by computing selectivity (probability) through division by the table cardinality m. Subsequently, to address the variable-length nature of queries—which are randomly sampled from either one or two columns as illustrated in Equation 1—we standardize their dimensionality to match the table's column count. This standardization is achieved by utilizing the infinity symbol to denote unspecified columns in the query, as formalized in Equation 2. Here, the original dataset table has dimensions (m, n), where P represents the Probability Density Function (PDF) and F represents the Cumulative Distribution Function (CDF), where each CDF corresponding to a query constraint.

$$\begin{aligned} F(3,\infty) = P(x \leq 3, y \leq \infty) = |\sigma_{col_0} \leq 3\,\&\,\sigma_{col_1} \leq \infty| = 12/m \\ F(3,5) = P(x \leq 3, y \leq 5) = |\sigma_{col_0} \leq 3\,\&\,\sigma_{col_1} \leq 5| = 10/m \end{aligned} \tag{2}$$

Furthermore, as the CDF values now have a uniform length of n, we stack them along with their selectivity vertically, resulting in a matrix (nested $2D$ array) with shape $(query\_size, n+1)$, as shown in Equation 3. Finally, given $F(3,\infty) = F(3, max\_2)$, we replace the infinity symbol in the Right Hand Side of the matrix in Equation 3 with the maximum value of each respective column (max_i).

$$\left[\begin{array}{cc|c} 3 & \infty & F(3,\infty) \\ 3 & 5 & F(3,5) \\ \infty & 9 & F(\infty,9) \\ \vdots & \vdots & \vdots \end{array}\right] = \left[\begin{array}{cc|c} 3 & max\_2 & 12/m \\ 3 & 5 & 10/m \\ max\_1 & 9 & 30/m \\ \vdots & \vdots & \vdots \end{array}\right] \tag{3}$$

Thus, the right-hand side (RHS) is the final result of preprocessing. With the infinity symbol replaced and a uniform format established, the RHS now consists entirely of Float data type and has a shape of $(query\_size, n+1)$, where each row represents a query.

3.3 GCN Model Architecture

In this work, we employ a Graph Convolutional Neural Network (GCN) [6] architecture to model the cumulative distribution function (CDF) within a multi-dimensional query space. A simple graph overview is shown in Figure 1. Each node in the graph uses its n-dimensional coordinate as the node feature, which represent a cumulative probability as Equation 4 where d is the dimension of query space and $x = [x_1, x_2, \cdots, x_d]$ is a point of multi-dimensional space D.

$$F_X(x) = P(X_1 \leq x_1, X_2 \leq x_2, \cdots, X_d \leq x_d) \tag{4}$$

The structural similarity between Equation 2 and Equation 4 enables direct integration of the first n columns from matrix Equation 3 into the graph structure as model inputs, while the final column provides ground truth selectivity values as training labels. This formulation demonstrates the natural compatibility of our GCN architecture with CDF learning, as it inherently aligns with the query set structure. The nodes participating in the query set are color-coded to represent their respective selectivity values. Additionally, the graph's directed structure ensures that node activations maintain the monotonic increasing property inherent to selectivity values, as specified in Figure 1.

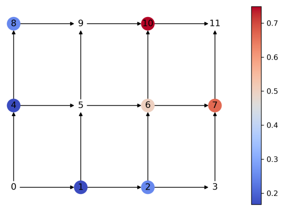

Figure 1: Graph of the GCN model, with node $(x_i, y_i) = P(X \leq x_i, Y \leq y_i)$ is monotonically increasing along each dimension.

The training process is accompanied by the update of the graph. As shown in Figure 2, the architecture consists of multiple graph convolution layers (e.g., 3 layers), where each node aggregates and transforms the information from their parent nodes through directed edges. Then a Sigmoid Linear Unit (SiLU) activation would

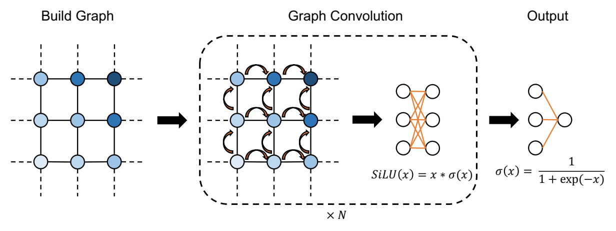

Figure 2: Layer structure of the GCN model.

be applied. The final sigmoid activation layer ensures the output values (query selectivity) are constrained within the range of 0 to 1.

As the query size increases, our model's graph structure and convolutional layer expand accordingly. Typically, the number of trainable parameters ranges from 105 to 177 for 2D models, while 625 to 897 for 3D models. Although these parameter counts are relatively small, further increases in graph dimensionality could potentially result in exponential growth of the parameter space.

We employ the mean squared error (MSE) to quantify the loss between model predicted selectivity and the true selectivity. Recognizing that the queries do not traverse every node within the graph, we apply masking to isolate and focus solely on the nodes involved in the queries, thus semi-supervised training method.

4 EXPERIMENTS

4.1 Preparation

Our datasets, the preprocessing matrix Equation 3, has a shape of $(query\_size, n+1)$, we then split it into training and test sets with a ratio of 0.8. Consequently, the training set has a shape of $(0.8 \times query\_size, n+1)$, while the test set has a shape of $(0.2 \times query\_size, n+1)$. The preprocessing method are adaptable to any table shape, and we follow the same pipeline as we conduct experiments on different datasets. Thus $query\_size$ ranges from 1 to 10^6; $n = 2$ for 2D-problem, and $n = 3$ for 3D-problem; $m = 48K$ for Census datasets and $m = 6.5K$ for Wine datasets. For 3D visulization, we assign the three selected features to the x, y, z-axis in the plots. For the 2D-visualization, we only use the x, y-axis features.

Throughout the training process, we use Adam as the optimizer to test hyperparameters in Table 2 and continuously monitor the loss and save model checkpoints whenever a new lowest loss is achieved, enabling us to retain the best-performing model parameters for optimal results.

Table 2: Hyperparameters of the GCN model.

Hyperparameter	Value
# of hidden layers	{1,2,3}
# of hidden channels	{4,8,16}
Epochs	{1000,3000,5000}
Batch sizes	{10^2, 10^4, 10^6}
Learning rate	{1e-2,1e-3,1e-4}

4.2 Evaluation Metrics

After the model is trained, a synthetic database could be built given the distribution learned by our model. To evaluate the synthetic database built from our model, we calculate the *Q-Error* using Equation 5 for both train and test query sets.

$$Q\text{-}Error = \frac{\max(\hat{C}, C)}{\min(\hat{C}, C)} \in [1, +\infty) \tag{5}$$

Where:

- C is the actual cardinality (the true number of records returned by the query);
- $\hat{C}$ is the estimated cardinality;

Q-Error measures how well the synthetic database satisfies the given constraints. It equals to 1 when the estimate is exactly correct. Q-Error serves as a widely adopted metric in database evaluation, providing an intuitive measure of the ratio between estimated and actual cardinalities in database tables. However, this metric has inherent limitations, particularly when dealing with small query sets, where multiple distributions may simultaneously satisfy the given constraints, making it challenging to identify the distribution closest to the ground truth. To address this limitation, we additionally employ *cross-entropy* in Equation 6 to quantify the similarity between our model-generated distribution and the actual data distribution.

$$H(P, Q) = -\sum_i P(x_i) \log Q(x_i) \tag{6}$$

Where:

- $P(x_i)$ is the true probability distribution of event x_i,
- $Q(x_i)$ is the predicted probability distribution of event x_i.

Cross-Entropy quantifies the statistical divergence between the generated instance and the original. A lower cross entropy indicates a closer alignment between the two database instances. In order to evaluate the computational complexity and scalability of the model, we also record the time required to run the model at the same time.

4.3 Baseline Models

We employ two established algorithms as our baselines. The first baseline utilizes the $Z3$ Satisfiability Modulo Theories (SMT) solver [4], which implements a symbolic constraint satisfaction approach through logical reasoning. In our SMT configuration, we formulate the cardinality constraint as a soft constraint with a unit penalty, enabling the solver to generate feasible solutions even when dealing with high-dimensional variable spaces. This formulation provides the solver with sufficient flexibility to address computational scalability challenges.

Our second baseline leverages Probabilistic Graphical Models (PGMs) [1], which constructs a generative distribution of the database that satisfies multiple cardinality constraints. We implement the PGM algorithm using the Sequential Least Squares Programming (SLSQP) optimizer available in the SciPy package[10], which provides robust numerical optimization capabilities for nonlinear optimization problems. In both baseline setups, a three-hour timeout is applied during evaluation to avoid excessive run times.

4.4 Evaluations

4.4.1 Training Error. We begin our analysis by examining the training error of the GCN model on the Census dataset (results for the Wine dataset are presented in the Appendix). Figure 3 illustrates the comparative analysis between the model-learned distribution and the ground truth distribution for queries of size 1000, while Figure 6 presents the corresponding results for the 3D Census dataset.

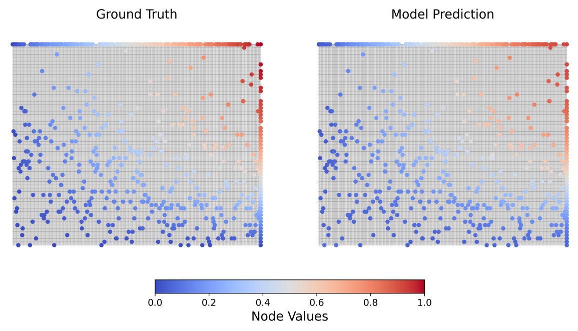

Figure 3: 2D visualization: Distribution learned by GCN (right) compared to the ground truth (left), ascending along both axes.

For query sizes less than or equal to 10^3, our GCN model achieves consistently low training Mean Squared Error (MSE), with values below 0.001 and reaching a minimum of 0.00046. Given that the data is normalized to the range $[0, 1]$, these results indicate an average prediction error of less than 3%, demonstrating strong model fit to the training data. However, when the query size increases to 10^4 or beyond, the training MSE plateaus at approximately 0.18. This performance degradation can be attributed to the increasing sparsity of the high-dimensional parameter space as both query size and model complexity grow, making convergence to the global optimum substantially more challenging. These optimization difficulties manifest in notable error margins in the final solutions.

It is worth noting that comparable MSE evaluations for the baseline models ($Z3$ solver and SLSQP optimizer) are not available, as these methods only return results upon successful convergence to a solution that satisfies all constraints, rather than providing intermediate or approximate solutions during the optimization process.

4.4.2 Q-error. As illustrated in Table 3, while both SMT and PGM approaches achieve near-optimal performance with Q-Error values approaching 1 for small query sizes (10 and 100), they fail to converge within the three-hour computational limit for queries of size 1000 or larger. Although these traditional methods demonstrate strong performance in scenarios with limited constraints, they exhibit significant scalability limitations when confronted with larger constraint space. However, the ability to handle a larger number of constraints is crucial for solving real-world problems, because incorporating more constraints increases the likelihood of approximating the ground truth, provided that the problem remains solvable. This argument is confirmed in our evaluation of cross-entropy (Section 4.4.4).

In contrast, our GCN-based approach demonstrates robust performance for queries up to size 10^6 in both training and testing query set. When evaluated on the test set with limited query sizes (10, 100), our model exhibits elevated errors, indicating suboptimal generalization to unseen data under sparse constraint conditions. However, this limitation is of minimal practical concern, as real-world applications typically involve substantially larger query sets. The model's superior performance on larger, more representative query sizes aligns well with practical deployment scenarios.

Note that for query sizes of 10^4 or larger, the Q-Error exhibits an increasing trend with the number of queries. We attribute it to the inherent limitations of our current GCN architecture in modeling large-scale query constraints. However, It is important to note here in the context of database generation, even Q-Error values exceeding 10 represent relatively modest deviations, such error margins are generally considered acceptable for practical applications. This scalability challenge is fundamentally rooted in the computational complexity of database generation from queries, which is an NP-hard problem. For large-scale query constraints, finding solutions that satisfy all constraints within a practical time frame is computationally intractable. This theoretical complexity barrier suggests that the observed performance degradation may be inherent to the problem rather than solely a limitation of our architectural choice.

Table 3: Q-Error on Census-2D: SMT and PGM use full queries, whereas GCN is trained on 80% of the queries and the remaining 20% are used for testing its generalization performance.

Model	Query	Median	75th	90th	Mean
SMT	10^1	1.0000	1.0000	1.0052	1.0052
	10^2	1.0000	1.0000	1.0000	1.0001
PGM	10^1	1.0000	1.0000	1.0054	1.0053
	10^2	1.0013	1.0032	1.0075	1.0358
GCN (train)	10^1	1.1880	1.3412	1.4546	1.2293
	10^2	1.7215	2.2244	3.5073	4.5222
	10^3	**1.4937**	**2.0283**	**3.9394**	**3.4498**
	10^4	2.6611	6.1961	16.444	9.6984
	10^5	2.5124	5.7128	16.053	14.608
	10^6	2.5018	5.7128	15.993	14.222
GCN (test)	10^1	1078.0	1616.5	1939.6	1078.0
	10^2	1.8586	3.0493	17.558	109.66
	10^3	**1.5270**	**2.0253**	**4.1350**	**2.7982**
	10^4	2.6098	6.0058	18.028	14.141
	10^5	2.5207	5.8214	16.601	13.866
	10^6	2.5229	5.7236	15.925	13.984

To potentially mitigate these challenges, future work could explore more sophisticated Graph Neural Network architectures, such as Graph Isomorphism Networks (GIN)[12] or Graph Attention Networks (GAT)[9], which might offer improved learning capabilities for complex constraint satisfaction problems.

We further validate our approach by presenting additional experimental results in the Appendix, including evaluations on Census-3D and both 2D- and 3D- configurations of the Wine dataset. These supplementary results demonstrate consistency with our primary findings, indicating that our model successfully generalizes across different datasets and dimensionalities. This comprehensive evaluation substantiates that our GCN-based approach offers a more scalable solution compared to conventional methods.

4.4.3 Running Time. GCN also surpasses baseline methods in terms of processing speed. We compare the run time of SMT and PGM with the training time of GCN (the training time for GCN directly corresponds to the time required to solve the CSP problem). SMT and PGM are implemented on Intel i9 CPU, as these traditional approaches are inherently sequential and cannot leverage parallel computing architectures. In contrast, our GCN model can effectively utilize the parallel processing capabilities of GPUs, and is run on a single NVIDIA H100 GPU.

Although SMT and PGM are efficient for solving small query sizes, they increase significantly with query size and fail to find out the solution within three hours' timeout when the query size reaches 1000. In contrast, the training time for GCN increases much more slowly than the baselines, only costing tens of seconds even when the query size reaches 1 million. Note that we consistently trained the model for 5000 epochs for every query size we experimented with, so the actual training time needed for GCN could be much lower than what is reported in Figure 4.

Meanwhile, the latency for model prediction with GCN is only around 0.001s when the query size reaches 1 million. This rapid inference capability indicates that once trained, our model can generate database instances with minimal computational overhead, making it particularly suitable for real-world applications or scenarios. Therefore, our GCN model demonstrates its potential as a scalable solution for database generation from query constraints.

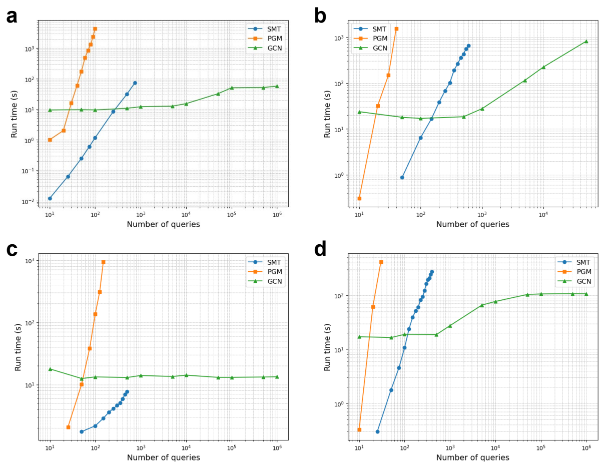

Figure 4: Processing time versus number of queries on a Census-2D, b Census-3D, c Wine-2D, d Wine-3D. Our model: GCN; Baseline model: PGM, SMT

4.4.4 Cross-Entropy. We further evaluated the model's performance in generating synthetic databases by calculating the cross entropy. Figure 5 demonstrates that, even the two baselines achieves near-optimal Q-Error performance in Table 3, their generated tables have notably higher cross entropy, indicating greater divergence from the true data distribution. This is because the constraint set is too small for these models to accurately approximate the ground truth. Moreover, the baseline shows limited improvement as the number of queries increases. In contrast, our GCN-based model maintains performance parity with the baseline for small query sizes, but achieves significantly lower cross entropy values when the query size exceeds 1,000.

This observation provides valuable insight into the relationship between constraint satisfaction and distribution matching. We employ Q-Error to evaluate how well the model-generated data satisfies the given constraints, while cross entropy measures the distributional similarity to the ground truth. Our analysis reveals that these metrics exhibit discordant behavior with small constraint sets but gradually converge as the constraint set expands. This phenomenon suggests that when constraints are sparse, multiple solutions may satisfy the constraints while having substantially different distributions. However, as the number of constraints grows, the solution space contracts, ultimately converging toward the true data distribution. This finding aligns with intuitive expectations about constraint satisfaction problems.

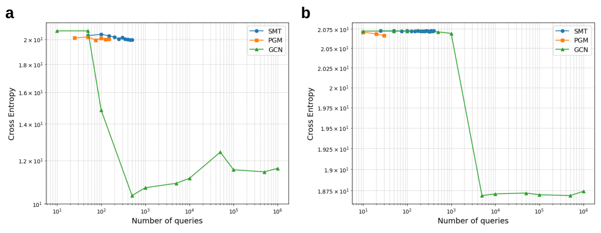

Figure 5: Cross entropy versus number of queries on a Census-3D and b Wine-3D.

5 CONCLUSIONS

From our experiments, both SMT- and PGM-based approaches encounter exponential complexity growth as the query size increases, indicating that they are only able to handle a small number of query constraints with low error. In contrast, our model demonstrates significant advancement by efficiently processing large-scale query sets across diverse datasets while maintaining both controllable error bounds and computational efficiency. These results highlight its potential as an effective approach in the field of synthetic database generation.

REFERENCES

[1] Arvind Arasu, Raghav Kaushik, and Jian Li. 2011. Data generation using declarative constraints. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 685–696. https://doi.org/10.1145/1989323.1989395

[2] Carsten Binnig, Donald Kossmann, Eric Lo, and M. Tamer Özsu. 2007. QAGen: generating query-aware test databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. Association for Computing Machinery, New York, NY, USA, 341–352. https://doi.org/10.1145/1247480.1247520

[3] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision support systems* 47, 4 (2009), 547–553. https://www.kaggle.com/datasets/rajyellow46/wine-quality

[4] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems*. https://api.semanticscholar.org/CorpusID:15912959

[5] Amir Gilad, Shweta Patwa, and Ashwin Machanavajjhala. 2021. Synthesizing Linked Data Under Cardinality and Integrity Constraints. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 619–631. https://doi.org/10.1145/3448016.3457242

[6] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[7] Eric Lo, Nick Cheng, and Wing-Kai Hon. 2010. Generating databases for query workloads. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 848–859. https://doi.org/10.14778/1920841.1920950

[8] Kolby Nottingham Markelle Kelly, Rachel Longjohn. [n.d.]. UCI Machine Learning Repository. https://www.kaggle.com/datasets/uciml/adult-census-income Accessed: 2024-09-26.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[11] Jiayi Wang, Chengliang Chai, Jiabin Liu, and Guoliang Li. 2021. FACE: a normalizing flow based cardinality estimator. *Proc. VLDB Endow.* 15, 1 (Sept. 2021), 72–84. https://doi.org/10.14778/3485450.3485458

[12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[13] Jingyi Yang, Peizhi Wu, Gao Cong, Tieying Zhang, and Xiao He. 2022. SAM: Database Generation from Query Workloads with Supervised Autoregressive Models. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1542–1555. https://doi.org/10.1145/3514221.3526168

APPENDIX

We provide the comparison of ground truth and predicted node values by GCN in Figure 6. The model is trained on Census-3D for visualization. The MSE loss is decreased to around 0.0009 on the training set.

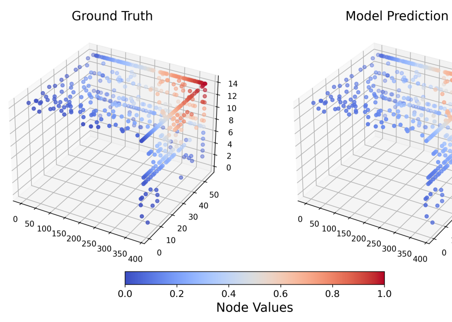

Figure 6: 3D CDF distribution of GCN model predictions (right) compared to ground truth CDF (left), ascending along both axes.

We summarize the Q-Error results on Census-3D (Table 4), Wine-2D (Table 5) and Wine-3D datasets (Table 6). Similar to observations for Census-2D, GCN model remains robust for query sizes over 10^3 where the baseline model fails to find the solution and achieves reasonable accuracy in the meantime.

Table 4: Q-Error on Census-3D: SMT and PGM use full queries, whereas GCN is only trained on 80% of the queries and the remaining 20% are used for testing its generalization performance.

Model	Query	Median	75th	90th	Mean
SMT	10^1	1.0340	1.2155	572.0603	571.9038
	10^2	1.0000	1.0308	1.1784	2.1677
PGM	10^1	1.2005	1.7610	2.6761	1.5646
	10^2	/	/	/	/
GCN (train)	10^1	2.7234	3.6430	3.9893	2.8527
	10^2	6.2615	8.2031	19.754	27.000
	10^3	**3.8858**	**5.6700**	**13.927**	**10.281**
	10^4	4.596	10.220	30.729	25.838
GCN (test)	10^1	56.465	82.232	97.693	56.465
	10^2	5.7325	7.2637	8.3241	6.3883
	10^3	**4.0602**	**6.3978**	**15.319**	**7.5972**
	10^4	4.7351	10.459	28.945	18.119

Table 5: Q-Error on Wine-2D: SMT and PGM use full queries, whereas GCN is only trained on 80% of the queries and the remaining 20% are used for testing its generalization performance.

Model	Query	Median	75th	90th	Mean
SMT	10^1	1.3077	2.7748	39.1238	10.1069
	10^2	1.1046	1.1867	1.3194	1.2114
PGM	10^1	1.4502	2.2570	9.6327	7.6497
	10^2	1.1206	1.0032	1.1938	1.2402
GCN (train)	10^1	1.0948	1.2081	1.9873	1.4134
	10^2	1.4740	1.6571	2.5245	7.2230
	10^3	1.9131	2.8824	6.0360	4.8958
	10^4	1.9576	3.3121	7.6255	6.5007
	10^5	1.8768	3.0470	6.4983	5.5147
	10^6	**1.8727**	**3.0169**	**6.4249**	**5.4407**
GCN (test)	10^1	336.65	504.32	604.93	336.65
	10^2	1.4795	1.6923	3.0038	14.907
	10^3	1.8898	3.2555	9.2614	5.6107
	10^4	1.9742	3.4045	7.7412	9.2515
	10^5	1.8655	3.0140	6.6393	5.5164
	10^6	**1.8732**	**3.0138**	**6.2645**	**5.3442**

Table 6: Q-Error on Wine-3D: SMT and PGM use full queries, whereas GCN is only trained on 80% of the queries and the remaining 20% are used for testing its generalization performance.

Model	Query	Median	75th	90th	Mean
SMT	10^1	1.3679	1.7120	30.5500	16.3567
	10^2	1.1470	1.2579	1.9913	1.6046
PGM	10^1	1.2412	1.7407	1.8975	1.4827
	10^2	/	/	/	/
GCN (train)	10^1	**4.5298**	**9.7927**	**47.794**	**21.337**
	10^2	10.618	16.216	48.002	34.167
	10^3	15.538	27.057	54.413	47.277
	10^4	37.361	58.475	105.59	71.760
	10^5	48.930	73.799	130.66	88.229
	10^6	48.965	73.829	129.93	88.069
GCN (test)	10^1	**3.0172**	**3.4408**	**3.6949**	**3.0172**
	10^2	8.5471	9.9991	12.015	9.4349
	10^3	15.623	29.032	58.115	33.108
	10^4	37.545	54.368	104.47	68.390
	10^5	49.024	73.799	129.91	87.134
	10^6	49.065	73.829	130.30	87.835