A PROOF OF THEOREM 1:

***Theorem 1**. Given a BN and its defined DAG $G = (V, E)$, representing a table T with attributes $V = \{T_1, \cdots T_n\}$, and a query $Q = (T'_1 = t'_1 \wedge \cdots \wedge T'_k = t'_k)$ where $T'_i \in V$. Let $G' = (V', E')$ be a sub-graph of G where V' equals $\bigcup_{1 \le i \le k} Ancestor(T'_i)$, and E' equals all edges in E with both endpoints in V'. $Ancestor(T'_i)$ includes all parent nodes of T'_i and all parents of parent node recursively. Then, performing* VE *of BN on full graph G is equivalent to running* VE *on reduced graph G'.*

Proof of Theorem 1: Given the probability query Q on original graph G and the reduced graph G' defined above, we define $Q_V = \{T'_1, \cdots, T'_k\}$ and $V/Q_V = T''_1, \cdots T''_{n-k}$. In this proof, we will only show that running *VE* on G is equivalent to running *VE* on G'. Then the proof for *progressive sampling* naturally follows as it is directly approximating the computation of *VE*.

First, recall that by the law of total probability, we have the following Equation 2.

$$P_T(T'_1 = t'_1, \cdots, T'_k = t'_k) = \sum_{t''_1 \in D(T''_1)} \cdots \sum_{t''_{n-k} \in D(T''_{n-k})} \left[\prod_{T'_i \in Q_V} P_T(T'_i = t'_i | Parents(T'_i)) * \prod_{T''_i \in V/Q_V} P_T(T''_i = t''_i | Parents(T''_i)) \right] \tag{2}$$

where $D(T''_i)$ denotes the domain of attribute T''_i and $Parents(T''_i)$ denotes the parents of node T''_i in graph G. For simplicity, here we refer to $Parents(T''_i)$ as $(T''_j = t''_j, \forall\, T''_j \in Parents(T'_i))$. The *VE* algorithm are essentially computing Equation 2 by summing out one attribute from V/Q_V at a time until all $T''_i \in V/Q_V$ are eliminated [27].

Alternatively, we can derive the following Equation 3 by the law of total probability and conditional independence assumption.

$$\begin{aligned} &P_T(T'_1 = t'_1, \cdots, T'_k = t'_k) \\ &= \sum_{T''_i \in \bigcup Parents(T'_j)_{1 \le j \le k}} \sum_{t''_i \in D(T''_i)} \left[P_T\left(T'_1 = t'_1, \cdots, T'_k = t'_k | \bigcup (Parents(T'_j)_{1 \le j \le k}\right) * P_T\left(\bigcup Parents(T'_j)_{1 \le j \le k}\right) \right] \\ &= \sum_{T''_i \in \bigcup Parents(T'_j)_{1 \le j \le k}} \sum_{t''_i \in D(T''_i)} \left[\prod_{T'_j \in Q_V} P_T(T'_j = t'_j | Parents(T'_j)) * P_T\left(\bigcup Parents(T'_j)_{1 \le j \le k}\right) \right] \end{aligned} \tag{3}$$

where $Parents(T''_i)$ denotes the parents of node T''_i in graph G, which is the same as parents of node T''_i in graph G'. By definition of reduced graph G' where $V' = \bigcup_{1 \le i \le k} Ancestor(T'_i)$. $Ancestor(T'_i)$ includes all parent nodes of T'_i and all parents of parent node recursively. Let $|V'| = n'$ and $V'/Q_V = T'''_1, \cdots, T'''_{n'-k}$. We can recursively write out $P_T\left(\bigcup Parents(T'_j)_{1 \le j \le k}\right)$ using Equation 3 and result in Equation 4.

$$P_T(T'_1 = t'_1, \cdots, T'_k = t'_k) = \sum_{t'''_1 \in D(T'''_1)} \cdots \sum_{t'''_{n'-k} \in D(T'''_{n-k})} \left[\prod_{T'_i \in Q_V} P_T(T'_i = t'_i | Parents(T'_i)) * \prod_{T'''_i \in V/Q_V} P_T(T'''_i = t'''_i | Parents(T'''_i)) \right] \tag{4}$$

Equation 4 has the same form as Equation 2 with less attributes in the summation. Thus the *VE* algorithm [27] can compute Equation 4 by eliminating one attribute from V'/Q_V at a time. Thus running *VE* on G is equivalent to running *VE* on G'.

B BAYESCARD PROGRESSIVE SAMPLING INFERENCE ALGORITHM

We present the pseudo code of BayesCard progressive sampling inference algorithm in Algorithm 1.

Algorithm 1 Progressive Sampling Inference Algorithm

Input: a table T with n attributes, a query Q with region R_Q and a PPL program defining the BN on P_T

```
Align the attributes in topological order T_1, ..., T_n
p ← 1, S ← [0]_{k×n}, an k × n dimension matrix of samples
for i ∈ {1, ..., n} do
    Take S[Par(T_i)], the columns in S corresponding to attributes in Par(T_i)
    P̂_i(T_i) ← (1/k) Σ_{d∈S[Par(T_i)]} P_T(T_i|d)
    p ← p * P̂_i(T_i ∈ R_Q(T_i))
    Define a PPL variable P'_i by normalizing P̂_i(t_i | t_i ∈ R_Q(T_i))
    S[i] ← k points sampled from P'_i
end for
return p
```

C BAYESCARD ENSEMBLE CONSTRUCTION ALGORITHM

We present the pseudo-code of BayesCard ensemble construction with budget algorithm in Algorithm 2.

Algorithm 2 BayesCard Ensemble Construction Algorithm

Input: a DB schema with n tables $T_1, \cdots, T_n$ and a budget k

Create the join tree $\mathcal{T} = (V, E)$ for the schema
Generate unbiased samples S for full outer join of the entire schema
Initialize a dependence matrix $M \in \mathbb{R}^{n \times n}$
for Each pair of tables $e = (T_i, T_j)$ **do**
  Calculate the RDC dependence level scores between all attributes in T_i and attributes in T_j
  $w_e \leftarrow$ average RDC scores
end for
if $k = 1$ **then**
  return $\mathcal{T}$ and learn a single PRM for each table
end if
for $k' \leftarrow 2, \cdots, k$ **do**
  Sort E in decreasing order based on w_e.
  for $e = (u, v) \in E$ **do**
    if u and v contain exactly k' tables in total **then**
      Update T by contracting nodes u, v to a single node $\{u, v\}$
    end if
  end for
end for
return $\mathcal{T}$ and learn a single PRM for each node in $\mathcal{T}$

D COMPUTING THE DEPENDENCE LEVEL BETWEEN TABLES

We use the randomized dependence coefficient (RDC) [35] as a measure of dependence level between two attributes. RDC is invariant with respect to marginal distribution transformations and has a low computational cost and it is widely used in many statistical methods [24, 59]. The complexity of RDC is roughly $O(n * log(n))$ where n is the sample size for the two attributes.

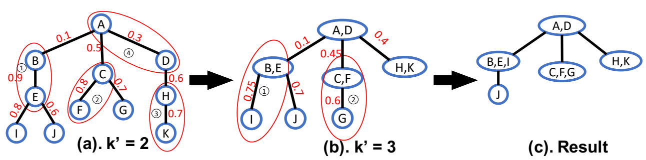

Figure 8: PRM Ensemble learning algorithm demonstration

D.1 Calculating the pairwise RDC score between two tables

Recall Figure 8, we have a DB schema with 11 tables $A, \cdots, K$ and their join tables are defined as a tree $\mathbb{T}$ on the left image. In addition, we have unbiased samples $\mathbb{S}$ of the full outer join of all tables in $\mathbb{T}$ using the previously mentioned approach [64]. Now consider $T, R \in A, \cdots, K$ as two random tables in this schema with attributes $T_1, \cdots, T_n$ and $R_1, \cdots, R_m$ respective. We can compute the pairwise RDC score between attributes T_i and R_j, RDC_{ij} based on $\mathbb{S}$, as described in [35]. Then we take the average as the level of dependence between T and R in the following Equation 5.

$$\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} RDC_{i,j} / (n * m) \tag{5}$$

Thus, we can compute the dependence level matrix M of size 11×11 with each entry specifying the dependence level between two tables in the schema. Then the edge weights of the original $\mathbb{T}$ on the left image can be directly taken from M. The complexity of calculating M is thus $O(m^2 * |\mathbb{S}| * log(|\mathbb{S}|))$ where m is the total number attributes in all tables.

D.2 Calculating the pairwise RDC score between two sets of tables

During the PRM ensemble construction procedure, we sometimes need to calculate the dependence level between two sets of tables, such as the dependence level of A, D and H, K as in the right image of Figure 8. Similar to the previous cases in Section D.1, this value can be directly computed from M.

Take $Att(T)$ denotes the set of attributes in table T. Same as Equation 5, the level of dependence between A, D and H, K is defined as Equation 7.

$$\begin{aligned}
&\sum_{ad \in Attr(\{A,D\})} \sum_{hk \in Attr(\{H,K\})} RDC_{ad,hk} / \\
&\quad \Big(|Attr(A) + Attr(D)| * |Attr(H) + Attr(K)| \Big) \\
&= \Bigg(\sum_{a \in Attr(A)} \sum_{h \in Attr(H)} RDC_{a,h} \\
&+ \sum_{a \in Attr(A)} \sum_{k \in Attr(K)} RDC_{a,k} \\
&+ \sum_{d \in Attr(D)} \sum_{h \in Attr(H)} RDC_{d,h} \\
&+ \sum_{d \in Attr(D)} \sum_{k \in Attr(K)} RDC_{d,k} \Bigg) \\
&\quad / \Big(|Attr(A) + Attr(D)| * |Attr(H) + Attr(K)| \Big) \\
&= \Big(M[A,H] * |Attr(A)| * |Attr(H)| \qquad (6) \\
&+ M[A,K] * |Attr(A)| * |Attr(K)| \\
&+ M[D,H] * |Attr(D)| * |Attr(H)| \\
&+ M[D,K] * |Attr(D)| * |Attr(K)| \Big) \\
&\quad / \Big(|Attr(A) + Attr(D)| * |Attr(H) + Attr(K)| \Big) \qquad (7)
\end{aligned}$$

Thus the weight of the edge can be updated quickly knowing the pre-computed M and the number of attributes in each table.

E ADDITIONAL EXPERIMENTAL RESULTS

E.1 GPU inference latency

Among all examined CardEst methods MSCN, Naru, and our BayesCard provide implementations specifically optimized on GPUs. We examine their query latency on a NVIDIA Tesla V100 SXM2 GPU with 64GB GPU memory. The comparison results on the dataset DMV and CENSUS are reported in Table 8.

We find that GPU can significantly speed up the computation and improve the latency of MSCN, Naru, and BayesCard.

Table 8: Comparison of CPU and GPU runtime.

Dataset	Method	Environment	Latency (ms)
DMV	MSCN	CPU	3.4
	MSCN	GPU	1.4
	Naru	CPU	86
	Naru	CPU	11
	BayesCard	CPU	2.1
	BayesCard	GPU	1.5
CENSUS	MSCN	CPU	4.8
	MSCN	GPU	2.6
	Naru	CPU	129
	Naru	CPU	18
	BayesCard	CPU	2.4
	BayesCard	GPU	1.7

E.2 Comparing with previous BN-base CardEst methods

BN-based CardEst methods have been explored decades ago for CardEst. Getoor et al. [17] used a *greedy* algorithm for BN structure learning, the variable elimination for probability inference, and referential integrity assumption for join estimation. Tzoumas et al. [54, 55] learned an exact-structured BN and used belief propagation for inference. Halford et al. [20] adopted the Chow-Liu tree structure learning algorithm, the VE inference algorithm, and the uniformity assumption for join estimation.

Apart from [20], the rest of the BN-based methods do not provide an open-source implementation. Therefore, we realize these methods in *BayesCard* and extend them to support IMDB datasets using the join uniformity assumption. Please note that the original paper extends BN to support multi-table join queries satisfying referential integrity [17], which is not satisfied in IMDB.

In table 9, we report and compare these methods with our *BayesCard* on both JOB-light and JOB-comp query workloads of IMDB datasets. We derive the following two findings. First, *BayesCard* is a unified Bayesian framework that subsumes these previously proposed BN-based CardEst algorithms. Second, *BayesCard* shows clear advantages over the previously proposed BN-based methods in terms of accuracy, latency, model size, and training time. This suggests the effectiveness of the proposed optimization in *BayesCard*.

Table 9: Performance of BN-based CardEst algorithms on IMDB datasets with two query workloads.

Workload	Method	50%	90%	95%	100%	Latency	Size	Train Time
JOB-light	[20]	2.16	28.0	74.6	306	35ms	309kb	16min
	[17]	2.31	24.8	105	466	162ms	1.8mb	25min
	[54, 55]	1.98	24.3	67.1	285	107ms	7.9mb	56min
	BayesCard	1.30	3.534	4.836	19.13	5.4ms	1.3mb	15min
JOB-Comp	[20]	2.21	25.6	2456	$7 \cdot 10^6$	53ms	309kb	16min
	[17]	2.19	47.2	2390	$7 \cdot 10^6$	207ms	1.8mb	25min
	[54, 55]	2.27	23.5	1804	$4 \cdot 10^6$	146ms	7.9mb	56min
	BayesCard	1.271	9.053	86.3	$4 \cdot 10^4$	6.2ms	1.3mb	15min

E.3 Experiments on SYNTHETIC datasets

The addition experiments on SYNTHETIC comparing *BayesCard* with other methods are reported in Table 10 and Table 11. Please note that we do not fine-tune the hyper-parameters of the DL-based methods since the training time on all the datasets takes so long that we can not afford to explore different hyper-parameters. But we believe that the experimental results are enough to show the insights. On the other hand, *BayesCard* does not have hyper-parameters to fine-tune, which is another advantage of our methods. We summarize our observations as follows.

Distribution (s): For the inference latency and model size, we find that increasing the Pareto distribution skewness would degrade the performance of *DeepDB* and *FLAT*, but has little effect on all other methods. This is because *DeepDB* and *FLAT* tend to generate larger models on more complex data. As a result, their training time also improves w.r.t. the skewness level. The training time of *Naru* and *MSCN* also grows w.r.t. skewness level, as their underlying DNNs need more time to model the complex distributions.

Correlation (c): For the inference latency and model size, the increase of c has a negative impact on *DeepDB* and *FLAT*, as their models become larger on more correlated data. However, *FLAT* behaves well on very highly correlated data since it can split them with other attributes to reduce the model size. The impact of c on *BayesCard* is mild, whose model size is still affordable. For the training time, the increase of c has impact on algorithms except *Histogram* and *Sampling*. This is also reasonable as they need more time to model the complex distribution. Note that, for each set of c, the training time of our *BayesCard* is still much less than them.

Domain (d): For the inference latency, model size and training time, the increase of d has significant impact on *Naru*, *DeepDB* and *FLAT*. This is increasing the number of attributes would increase the data complexity exponentially, so they need more neurons or nodes to model the distribution. Whereas, the impact on our *BayesCard* is much mild.

Scale (n): Similar to domain size, increasing the number of attributes also increases the data complexity exponentially, and thus we expect to see an increase in latency, model size, and training time for almost all methods. However, in comparison with *Naru*, *DeepDB*, *FLAT* and *MSCN*, the impact on *BayesCard* is not very significant.

Summary: *In comparison with DL-based CardEst methods, our BayesCard attains very stable and robust performance in terms of inference latency, model size, and training time.*

Table 10: Stability performance of different CardEst method w.r.t. changes in data distribution skewness and correlation.

CardEst Methods	Algorithm Criteria	Distribution Skewness (s) c=0.4, d=100, n=10						Attribute Correlation (c) s=1.0, d=100, n=10					
		s=0	s=0.3	s=0.6	s=1.0	s=1.5	s=2.0	c=0	c=0.2	c=0.4	c=0.6	c=0.8	c=1.0
BayesCard	Accuracy (95% q-error)	1.06	**1.09**	1.25	**1.49**	2.39	2.28	**1.32**	1.28	1.48	2.13	**1.49**	**1.00**
	Latency (ms)	3.5	2.8	3.7	2.4	2.2	3.0	**0.1**	2.9	2.4	1.6	3.3	2.1
	Model size (kb)	534	530	538	529	403	514	9.5	525	508	478	605	396
	Training time (s)	17.5	18.2	17.3	19.2	16.8	14.4	5.2	19.2	17.3	37.9	21.3	45.6
Histogram	Accuracy (95% q-error)	136	112	195	240	219	277	**1.32**	73.2	240	1403	$2 \cdot 10^4$	$9 \cdot 10^4$
	Latency (ms)	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
	Model size (kb)	**11.5**	**9.7**	**8.2**	**8.8**	**7.3**	**7.5**	**9.5**	**8.3**	**8.8**	**9.7**	**8.4**	9.1
	Training time (s)	**3.1**	**3.0**	**3.1**	**4.4**	**3.9**	**4.1**	**3.0**	**2.9**	**3.4**	**3.0**	**3.3**	**3.1**
Sampling	Accuracy (95% q-error)	**1.03**	1.67	2.87	2.21	55.9	142.1	1.78	1.62	2.21	4.07	2.10	1.03
	Latency (ms)	52	54	52	60	52	49	51	53	61	47	52	52
	Model size (kb)	-	-	-	-	-	-	-	-	-	-	-	-
	Training time (s)	-	-	-	-	-	-	-	-	-	-	-	-
Naru	Accuracy (95% q-error)	**1.03**	1.22	1.78	3.62	28.8	21.4	1.76	**1.23**	3.62	**2.09**	1.71	1.10
	Latency (ms)	58	67	58	62	60	66	73	65	62	61	66	59
	Model size (kb)	3670	3670	3670	3670	3670	3670	3670	3670	3670	3670	3670	3670
	Training time (s)	4460	4910	4879	5503	5908	6371	1702	4702	5503	9915	4962	1308
DeepDB	Accuracy (95% q-error)	4.51	4.89	15.1	14.0	14.2	19.0	1.32	3.58	15.1	117	663	108
	Latency (ms)	4.4	4.6	7.5	7.3	5.8	9.2	**0.1**	**4.3**	7.3	10.6	16.4	13.2
	Model size (kb)	701	597	834	1107	1104	1315	9.5	570	1198	1864	5532	1907
	Training time (s)	131	133	197	247	271	280	5.5	131	244.2	421	2570	830
FLAT	Accuracy (95% q-error)	1.06	1.15	**1.23**	1.76	**2.25**	**2.11**	1.32	1.27	1.76	2.11	1.73	**1.00**
	Latency (ms)	0.6	0.9	0.6	0.5	1.5	1.7	**0.1**	0.7	0.5	4.1	17.8	0.2
	Model size (kb)	76	101	80	75	430	580	9.5	103	75	1201	1889	**4.7**
	Training time (s)	91	93	127	142	240	253	5.5	133	244.2	629	1370	17.0
MSCN	Accuracy (95% q-error)	55.1	160	105	94	129	340	51.3	54.0	94	145	544	620
	Latency (ms)	1.1	1.1	1.0	1.1	1.3	1.0	1.0	1.4	1.2	1.0	1.3	1.1
	Model size (kb)	3200	3200	3200	3200	3200	3200	3200	3200	3200	3200	3200	3200
	Training time (s)	1203	1208	959	1430	871	1770	922	935	1432	955	1831	880

Table 11: Scalability performance of different CardEst method w.r.t. changes in data domain size and the number of attributes.

CardEst Methods	Algorithm Criteria	Domain Size (d) s=1.0, c=0.4, n=10						Number of Attributes (n) s=1.0, c=0.4, d=100					
		d=10	d=100	d=500	d=1000	d=5000	d=10000	n=2	n=5	n=10	n=50	n=100	n=200
BayesCard	Accuracy (95% q-error)	1.29	**1.49**	**1.05**	**1.04**	25.3	49.1	1.04	1.12	**1.49**	**2.58**	**1.97**	**3.02**
	Latency (ms)	1.0	2.4	2.8	11.5	2.2	3.6	0.4	1.5	2.4	4.7	11.3	15.1
	Model size (kb)	19.3	542	982	1280	168	417	64.2	236	542	2820	5400	$1 \cdot 10^4$
	Training time (s)	11.1	14.9	33.4	48.7	**7.3**	**6.5**	2.01	4.73	14.5	113	576	1907
Histogram	Accuracy (95% q-error)	15.2	240	498	1066	$2 \cdot 10^4$	$6 \cdot 10^4$	19.3	103	240	$3 \cdot 10^4$	$1 \cdot 10^5$	$9 \cdot 10^5$
	Latency (ms)	**0.1**	**0.1**	**0.2**	**0.4**	**0.8**	**0.1**	**0.1**	**0.1**	**0.1**	**0.5**	**0.4**	**0.7**
	Model size (kb)	**0.8**	**9.8**	**14.0**	**12.8**	**13.2**	**14.3**	**1.5**	**4.0**	**10.1**	**48.8**	**173**	**308**
	Training time (s)	**1.1**	**3.0**	**4.3**	**6.7**	11.5	15.0	**0.8**	**1.6**	**3.1**	**16.4**	**44.8**	**90.7**
Sampling	Accuracy (95% q-error)	1.95	2.21	2.09	4.93	**21.2**	57.9	1.04	1.26	2.21	342	$2 \cdot 10^4$	$3 \cdot 10^4$
	Latency (ms)	59	55	60	63	57	59	45	60	77	109	123	135
	Model size (kb)	-	-	-	-	-	-	-	-	-	-	-	-
	Training time (s)	-	-	-	-	-	-	-	-	-	-	-	-
Naru	Accuracy (95% q-error)	**1.02**	3.62	7.30	89.4	292	1783	1.39	**1.09**	3.62	121.0	475	1098
	Latency (ms)	32	62	81	115	154	260	33	37	62	225	473	726
	Model size (kb)	3140	3670	6135	8747	$2 \cdot 10^4$	$4 \cdot 10^4$	2447	3050	3670	6673	8010	$1 \cdot 10^4$
	Training time (s)	1032	5503	5607	6489	5980	$1 \cdot 10^4$	1157	4201	5503	6930	$1 \cdot 10^4$	$2 \cdot 10^4$
DeepDB	Accuracy (95% q-error)	1.08	10.0	15.1	107	61	213	1.04	1.17	15.1	257	1490	$1 \cdot 10^4$
	Latency (ms)	1.3	7.5	8.8	11.9	10.7	19.0	0.6	1.4	7.5	25.4	67.1	109
	Model size (kb)	29.5	834	1234	1910	1781	3974	35.0	129	834	6710	$3 \cdot 10^4$	$9 \cdot 10^4$
	Training time (s)	25	197	769	2310	4155	$1 \cdot 10^4$	21	65	197	3698	8930	$2 \cdot 10^4$
FLAT	Accuracy (95% q-error)	1.08	1.76	1.35	1.17	27.6	**44.0**	**1.02**	**1.09**	1.76	255	2015	$1 \cdot 10^4$
	Latency (ms)	0.5	0.6	1.5	18.0	15.9	49.7	0.4	0.5	0.5	25.9	66.0	110
	Model size (kb)	16.1	75.3	310	2701	1980	5732	15.0	49.9	75.3	6908	$3 \cdot 10^4$	$9 \cdot 10^4$
	Training time (s)	15.5	142	198	2670	1535	9721	9.7	48.6	142	4017	$1 \cdot 10^4$	$2 \cdot 10^4$
MSCN	Accuracy (95% q-error)	53.0	94.4	106	188	173	290	18.8	40.5	94.4	1783	8084	$3 \cdot 10^4$
	Latency (ms)	0.9	1.1	1.1	1.1	1.0	1.2	0.4	0.9	1.1	1.3	1.8	2.0
	Model size (kb)	3200	3200	3200	3200	3200	3200	2430	2871	3200	3328	3609	3827
	Training time (s)	1179	1430	1329	1398	1530	1230	719	821	1430	1600	2031	2299