PairwiseHist: Fast, Accurate and Space-Efficient Approximate Query Processing with Data Compression

Aaron Hurst
Aarhus University
Denmark
ah@ece.au.dk

Daniel E. Lucani
Aarhus University
Denmark
daniel.lucani@ece.au.dk

Qi Zhang
Aarhus University
Denmark
qz@ece.au.dk

ABSTRACT

Exponential growth in data collection is creating significant challenges for data storage and analytics latency. Approximate Query Processing (AQP) has long been touted as a solution for accelerating analytics on large datasets, however, there is still room for improvement across all key performance criteria. In this paper, we propose a novel histogram-based data synopsis called PairwiseHist that uses recursive hypothesis testing to ensure accurate histograms and can be built on top of data compressed using Generalized Deduplication (GD). We thus show that GD data compression can contribute to AQP. Compared to state-of-the-art AQP approaches, PairwiseHist achieves better performance across all key metrics, including 2.6× higher accuracy, 3.5× lower latency, 24× smaller synopses and 1.5–4× faster construction time.

PVLDB Reference Format:
Aaron Hurst, Daniel E. Lucani, and Qi Zhang. PairwiseHist: Fast, Accurate and Space-Efficient Approximate Query Processing with Data Compression. PVLDB, 17(6): 1432 - 1445, 2024.
doi:10.14778/3648160.3648181

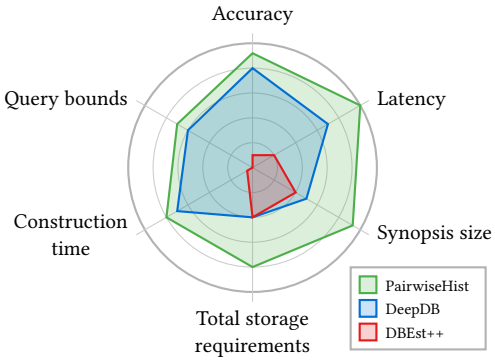

Figure 1: Relative performance comparison of PairwiseHist, DeepDB and DBEst++ summarising Figs. 8, 10 and 11 and Table 6. Outer rings indicate better performance. Each interval represents approximately 2× improvement.

1 INTRODUCTION

Rapidly advancing digital transformation is driving exponential growth in data volumes across many sectors. This poses significant challenges for current infrastructure and data management solutions, which must not only handle swelling data workloads, but also meet the demands of increasingly advanced analytics. Efficient data storage and analytics techniques are therefore crucial.

Approximate Query Processing (AQP) is a well-established field that focuses on enabling fast analytics on Big Data by sacrificing some degree of accuracy [5, 30]. AQP techniques are typically based on either sampling or data synopses, or a hybrid of the two [30, 32, 41, 42]. In general, small samples or compact synopses enable analytics to be performed over large datasets within required latency constraints. However, all AQP approaches exhibit a distinct trade-off between accuracy, latency and synopsis size.

In this paper, we propose a novel histogram-based data synopsis for AQP called PairwiseHist that consists of three key components: i) one-dimensional histograms that capture within-column data distributions, ii) two-dimensional histograms that capture relationships between each pair of columns (hence, *Pairwise*Hist), and iii) small metadata for each histogram bin that enhance query precision, including the minimum, maximum and number of unique values. By using histograms, PairwiseHist inherits desirable properties including accurate query bounds and effective outlier recall, while the small number of histograms minimises synopsis size. All histograms within PairwiseHist are constructed using recursive hypothesis testing that ensures each bin contains uniformly distributed data, leading to high query accuracy. Low query execution latency is realised by novel methods for resolving multi-predicate queries that require only a few relatively small matrix multiplications. Overall, PairwiseHist delivers all-round superior performance across accuracy, latency, synopsis size, synopsis construction time and query bounds compared to state-of-the-art AQP techniques, as illustrated in Fig. 1.

PairwiseHist takes inspiration from recent works in data compression that demonstrate (approximate) data clustering can be performed directly on compressed data without decompression [22–24]. That is, by using Generalized Deduplication (GD) data compression [54–57], part of the compressed data known as *bases* can serve as a data synopsis on which analytics tasks can be performed efficiently. Compared to GD bases, PairwiseHist significantly reduces storage requirements and improves accuracy by using low-dimensional histograms with variable precision.

While PairwiseHist is a stand-alone AQP technique in its own right, we also propose implementing it alongside data compression, as illustrated in Fig. 2. Specifically, we use GreedyGD [23], a recent version of GD, which is a lossless data compression algorithm that offers state-of-the-art compression ratios and low random access

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648181

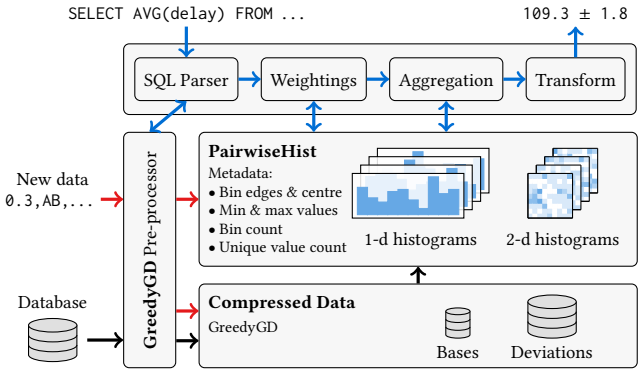

Figure 2: Our proposed AQP framework with compression, including data ingestion and PairwiseHist construction (black arrows), query execution (blue) and data updates (red).

cost. While originally designed for IoT, GD has also proved effective in several other domains [14, 16, 23, 40, 47]. In our proposed AQP framework, GreedyGD both reduces overall storage requirements and accelerates synopsis construction.

Due to its small synopsis size and low query latency, our approach also enables Edge analytics, even on resource-constrained devices. This comes with many benefits, including low (communication) latency, scalability, privacy, energy savings and mobility [26, 39, 60]. Meanwhile, PairwiseHist's fast construction and low storage requirements reduce Cloud storage and computing costs and enable more frequent updates.

In summary, the key contributions of this paper include:

(1) A novel histogram-based AQP technique called PairwiseHist that uses a combination of one- and two-dimensional histograms and an efficient storage encoding to deliver data synopses that are 24× smaller and up to 4× faster to construct than state-of-the-art AQP methods DeepDB [19] and DBEst++ [34],
(2) Query execution techniques for seven common aggregation functions that deliver at least 2.6× better accuracy (0.28% median error vs. 0.73–28.9%), 3.5× lower latency and more accurate bounds than the state-of-the-art,
(3) A novel AQP framework that integrates data compression to reduce overall storage requirements by up to 4.3×, and
(4) A comprehensive performance evaluation of PairwiseHist in comparison to the state-of-the-art across 11 real-world datasets and two datasets scaled-up using IDEBench [12].

The remainder of this paper is structured as follows. Section 2 describes related work. Section 3 provides our system overview. Section 4 outlines the PairwiseHist data structure and construction algorithm. Section 5 defines the theory and mathematical formulations for query execution. Section 6 evaluates the performance of PairwiseHist. Finally, Section 7 concludes the paper.

2 RELATED WORK

AQP methods are typically classified as either sampling-based or synopses-based [5]. Sampling approaches can be either offline or online [8]. Online methods select samples at query time and attempt to minimise the amount of data that must be accessed to achieve a desired accuracy. Gapprox [4], for example, uses cluster sampling to reduce processing. Offline approaches, on the other hand, attempt to prepare in advance the most representative sample for the most queries. For example, Babcock et al. [6] use biased sampling and BlinkDB [3] uses stratified sampling. More recent offline sampling work has focused on additional features, such as integration middleware [41] and increasing sample re-usability [45]. However, sampling methods generally struggle with skewed data and may support limited aggregation functions [30].

Synopsis-based approaches build a compact summary of the data using statistical or machine learning techniques. Histograms are a classical synopsis approach that is widely used in selectivity estimation, which involves estimating the number of tuples that a query will access. This is an important step in database query optimisers and is equivalent to AQP for COUNT queries. Many histogram algorithms exist, including simple equi-width and equi-depth histograms, as well as advanced methods, such as V-optimal histograms [25], entropy-based histograms [51] and others [1, 11, 50]. Most approaches, however, are limited to one dimension. Multi-dimensional histograms are notoriously challenging to construct and their storage scales exponentially with the number of dimensions [10, 62]. Nonetheless, some multi-dimensional histograms for selectivity estimation are available, including DigitHist [48], DMMH [62] and STHoles [7]. To avoid the pitfalls of high dimensionality, DigitHist uses lossy compression, DMMH uses density modelling and STHoles uses previous query results. Cormode et al. [10] also discuss using collections of histograms, which is the approach we have taken with PairwiseHist. Common to all histograms is that accuracy depends on ensuring a uniform distribution of tuples within each bin. To the best of our knowledge, our approach is the first to utilise hypothesis testing for this purpose.

Most recent synopsis-based AQP works focus on machine learning. These can be classified as either generative, which create synthetic data for evaluating queries, or inferential, which directly predict AQP results [29]. In [27, 28], a generative model that captures the joint probability distribution of a database using Mixed Sum-Product Networks (MSPNs) is proposed. This approach can evaluate simple queries directly using the MSPN weights or generate (synthetic) samples to answer more complex queries. While this provides high accuracy, low latency and compact summaries, it requires significant construction time (hours for just 10^6 samples).

DeepDB [19] proposes Relational Sum Product Networks (RSPNs), which are purely inferential and extend (M)SPNs to support multiple database tables, complex queries and model updates. In comparison to [28], DeepDB achieves much faster construction, but has poor latency with multi-predicate queries, higher storage requirements and can give imprecise bounds. Our evaluation also revealed that DeepDB does not support OR relationships between predicates, despite claiming to, and performs poorly on real-world data.

DBEst [35] is an inferential approach that uses a combination of model types. Kernel density estimators model individual columns, while regression models capture relationships between pairs of columns. This is similar to PairwiseHist with the density and regression models corresponding to one- and two-dimensional histograms, respectively. A significant limitation of DBEst is that a new

model is required for every query template, which limits flexibility and exponentially increases synopsis storage requirements.

DBEst++ [34] improves DBEst by switching to mixture density networks for both regression and density modelling. This significantly reduces storage requirements, delivers better accuracy and latency, and supports data updates. However, storage requirements for DBEst++ are misleading due to each model template requiring its own model. Our tests also revealed that DBEst++ does not support queries involving more than two columns, OR relationships between predicates, queries on only categorical columns or inequality predicates on date/time columns.

Other machine learning approaches include LAQP [61], which combines an error prediction model with sampling, Electra [49], which focuses on queries with many-predicates, NeuroSketch [58], which is a bounded inferential model trained on queries, and Generative Adversarial Networks [13]. Typically, machine learning methods are constructed from samples and exhibit similar limitations as sampling-based AQP methods, namely vulnerability to skewed data and limited aggregation function support.

A small number of unified approaches that combine online sampling and synopses have also been proposed. For example, AQP++ [42] generates a set of pre-computed aggregations known as a prefix cube and answers queries by supplementing relevant prefix cube elements with sampling. A similar approach has also been proposed for selectivity estimation [38]. More recently, [32] proposed PASS, which has a more flexible synopsis design, reminiscent of DigitHist [48], that consists of a hierarchical set of pre-computed aggregates at different resolutions that guide online stratified sampling. Unlike other sampling-based approaches, PASS provides both probabilistic and deterministic bounds. However, a significant limitation is nearly 30× slower construction than AQP++ [32], which itself requires over 20 minutes for just a 50 MB sample [42].

Many AQP approaches provide query error bounds, which give analysts an indication of the confidence that they can place in AQP results. Bounds are typically based on probabilistic confidence intervals [30, 59], but can also be deterministic [3, 4, 31]. Unfortunately, probabilistic bounds can often be incorrect [2] while deterministic bounds may be too broad to be useful.

The overall performance of state-of-the-art AQP techniques is summarised in Table 1, in which versatility refers to the variety of supported query templates. As can be seen, PairwiseHist delivers comprehensively superior performance. Moreover, by using data compression, it uniquely offers significant overall storage reduction.

Table 1: PairwiseHist compared to previous AQP works.

	Accuracy	Latency	Bounds	Size	Build	Versatility
PairwiseHist	**<1%**	**sub-ms**	**yes**	**sub-MB**	**secs**	**v. high**
VerdictDB [41]	1%	seconds	**yes**	GBs	?	**v. high**
Gapprox [4]	<5%	seconds	**yes**	n/a	n/a	low
BlinkDB [3]	<10 %	seconds	**yes**	GBs	n/a	high
DigitHist [48]	1%	**sub-ms**	**yes**	MBs	mins	v. low
DMMH [62]	1–2%	ms	no	**sub-MB**	**secs**	v. low
STHoles [7]	10%	?	no	**sub-MB**	?	v. low
DeepDB [19]	1%	ms	**yes**	MBs	mins	high
DBEst++ [34]	1%*	ms	no	MBs	hours	low
NeuroSketch [58]	5%	**sub-ms**	**yes**	**sub-MB**	mins	**v. high**
LAQP [61]	10%	ms	no	**sub-MB**	?	**v. high**
Electra [49]	10%	?	no	?	?	low
PASS [32]	**<1%**	ms	**yes**	MBs	mins	high
AQP++ [42]	**<1%**	seconds	**yes**	MBs	mins	high

"?" indicates not reported by the authors. *Much larger error observed in practice.

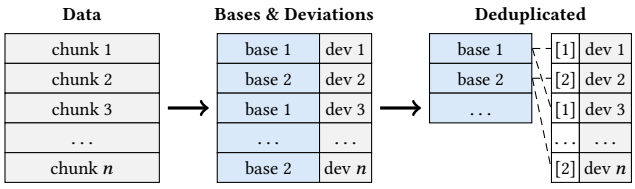

Figure 3: GD splits data into bases and deviations.

3 SYSTEM OVERVIEW

Problem Definition. Consider a dataset $\mathcal{D}$ with N rows and d attributes $X_1, \ldots, X_d$ and queries of the form:

SELECT $F(X_i)$ FROM $\mathcal{D}$ WHERE P_1 AND/OR $P_2 \ldots$
GROUP BY $\ldots$;

where F is an aggregation function (e.g. AVG), $P_1, P_2, \ldots$ are predicate conditions of the form "X_j OP LITERAL", where OP is a binary logical operator (i.e., $<, >, \leq, \geq, =$ or $\neq$) and LITERAL is a valid value for column X_j, and GROUP BY can be applied to any categorical column. It is assumed that $\mathcal{D}$ is large enough such that exact query execution is prohibitively expensive. Therefore, the task is to design a framework such that bounded approximate query results can be obtained with high accuracy and low latency, while minimising synopsis size, synopsis construction time and overall storage requirements. Missing values must also be supported.

Data Compression. In our proposed AQP framework, PairwiseHist is applied on top of compressed data, which reduces storage requirements. As shown in Fig. 2, GreedyGD compresses incoming data, which includes pre-processing to improve compressability. Pre-processing is applied to each column independently based on its data type and includes minimum value subtraction, floating point to integer conversion (e.g. 10.22 to 1022), frequency-ranked categorical value encoding (i.e., most common encoded as 0, second most as 1, etc.) and encoding missing values. Importantly, pre-processing does not require additional storage or memory and datasets can be processed in arbitrarily-sized batches, which allows processing of datasets that are too large to fit in memory.

GreedyGD splits data *chunks* into *bases* and *deviations*. Bases contain the majority of the information and are deduplicated, while deviations are stored verbatim with IDs linking them to the appropriate bases, as illustrated in Fig. 3. Compression is achieved when there are few bases compared to the number of data chunks. In our framework, a chunk corresponds to a row in a relational database table and bases contain the most significant bits from each attribute. New rows can be added incrementally to the compressed data.

PairwiseHist. Once the data is compressed, PairwiseHist is built on top of the compressed data by taking (a sample of) the bases as input for the initial histogram bin edges. The histograms are then refined using hypothesis testing to ensure that the distributions

Table 2: Notations

Construction	d	No. columns in the dataset
	N	No. rows in the dataset
	N_s	No. samples used to construct PairwiseHist
	ρ	PairwiseHist sampling ratio, N_s/N
	M	Minimum points for bins to be split
	α	Significance level for hypothesis tests
Histograms	$k^{(i)}$	No. histogram bins in column i
	$\boldsymbol{e}^{(i)}$	Bin edges for column i
	$\boldsymbol{H}^{(i)}$	Bin counts matrix for column i
	$\boldsymbol{v}^{(i)\pm}$	Bin minimum and maximum values for column i
	$\boldsymbol{c}^{(i)}$	Bin midpoints for column i
	$\boldsymbol{c}^{(i)\pm}$	Bin weighted centre bounds for column i
	$\boldsymbol{u}^{(i)}$	No. unique values in each bin for column i
Hypothesis	Δ	Bin width
	δ	Sub-bin width
	s	No. sub-bins in a given bin
	$\hbar$	Sub-bin count
Queries	$\boldsymbol{\beta}$	Bin coverage
	$\boldsymbol{w}$	Bin weightings
	a	No. sub-bins fully covered by a query
	b	No. sub-bins fully or partially covered by a query
	t^*	Bin index of query result (MEDIAN/MIN/MAX only)

of tuples within individual bins are approximately uniform. This process is described in detail in Section 4. As illustrated in Fig. 2, PairwiseHist consists of one-dimensional histograms for each column, two-dimensional histograms for every pair of columns and various metadata for each histogram bin. This collection of histograms enables rapid query aggregation on any column subject to arbitrary combinations of predicate conditions. Furthermore, compared to a naïve multi-dimensional histogram, for which storage increases exponentially with the number of dimensions [62], our approach reduces storage requirements from $O(k^d)$ to $O(d^2k^2)$ for d dimensions and k bins per dimension. A compact storage encoding for PairwiseHist is also proposed in Subsection 4.3.

PairwiseHist can also support multi-table databases. That is, queries across different tables can be resolved via two-dimensional histograms involving the primary/foreign keys. However, in this paper we focus on single-table queries.

If PairwiseHist is used independently of GreedyGD, then histograms are constructed from scratch (i.e., without using bases as initial bin edges), with slightly longer synopsis construction time due to less precise initial conditions.

4 PAIRWISEHIST

PairwiseHist generates its collection of histograms by recursively refining (i.e., splitting) an initial set of histogram bins using hypothesis testing, which ensures that data points within individual bins are sufficiently uniformly distributed. Refinement is terminated when a bin is either uniform or has too few points to be split further. PairwiseHist is thus parameterised by the minimum number of points required for a bin to be split, M, the hypothesis

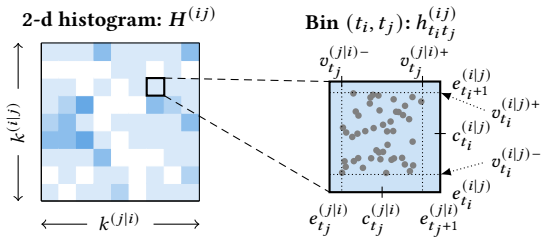

Figure 4: Notation for two-dimensional histograms with bin counts matrix $H^{(ij)}$ and individual bin count $h^{(ij)}_{t_i t_j}$.

test significance level, α, and the number of samples used to construct PairwiseHist, N_s. Before describing PairwiseHist in detail, this section provides an overview of the key notation, which is summarised in Table 2.

Bin edges for one-dimensional histograms are denoted by the vector $\boldsymbol{e}^{(i)} = \left\langle e^{(i)}_1, e^{(i)}_2, \ldots, e^{(i)}_{k^{(i)}} \right\rangle$, where the superscript (i) indicates the column to which the histogram applies and $k^{(i)}$ denotes the number of bins for the one-dimensional histogram for column i. For two-dimensional histograms, the bin edges for columns i and j are denoted $\boldsymbol{e}^{(i|j)}$ and $\boldsymbol{e}^{(j|i)}$, respectively. This superscript notation for two-dimensional histograms, $(i|j)$ and $(j|i)$, indicates the fact that two-dimensional histograms may have additional bin edges in either or both dimensions compared to one-dimensional histograms due to additional refinement. One-dimensional histogram bin counts are denoted by the $k^{(i)} \times k^{(i)}$ diagonal matrix $\boldsymbol{H}^{(i)}$ where the diagonal elements contain the bin counts. That is:

$$\boldsymbol{H}^{(i)} = \begin{bmatrix} h^{(i)}_1 & 0 & \cdots & 0 \\ 0 & h^{(i)}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & h^{(i)}_{k^{(i)}} \end{bmatrix} \tag{1}$$

where $h^{(i)}_t$ denotes the bin count for bin t. Two-dimensional histogram bin counts are similarly denoted by the $k^{(i|j)} \times k^{(j|i)}$ matrix $\boldsymbol{H}^{(ij)}$. In this case, off-diagonal values may be non-zero.

PairwiseHist stores various metadata for each bin, namely: 1) *minimum* and *maximum* actual data values, $\boldsymbol{v}^{(i)-}_t$ and $\boldsymbol{v}^{(i)+}_t$; 2) the bin *midpoint*, $c^{(i)}_t$, which is equidistant between the minimum and maximum values; 3) bounds on the weighted centre of the data points within the bin, $c^{(i)-}_t$ and $c^{(i)+}_t$; and 4) the *unique count*, $u^{(i)}_t$, which is the number of unique values in the bin. Note that for two-dimensional histograms, each of these metadata apply to both dimensions, as illustrated in Fig. 4.

In this paper, vectors are indicated by bold lower case letters and matrices by bold upper case letters. The L1 norm is denoted by $\|\cdot\|_1$, the vector dot product by $\boldsymbol{x} \cdot \boldsymbol{y}$, Hadamard (element-wise) multiplication by $\boldsymbol{x} \odot \boldsymbol{y}$ and Hadamard division by $\boldsymbol{x} \oslash \boldsymbol{y}$. We also use specific index variables for specific purposes. That is, i and j are used for column indices within a dataset, t is used for histogram bins and r is used for sub-bins. The terms *column* and *dimension* are used interchangeably, as well as *tuple* and *data point*.

Algorithm 1 BuildPairwiseHist

Inputs: dataset $\mathcal{D}$, sample size N_s, minimum points M, hypothesis test significance α, initial bin edges E_0 (optional)

Outputs: PairwiseHist data structure

1: $\boldsymbol{D} \leftarrow$ downsample $\mathcal{D}$ to N_s rows
2: **for** $i = 1, 2, \ldots,$ number of columns in $\mathcal{D}$ **do**
3: *// 1-d histograms*
4: $\hat{\boldsymbol{e}}^{(i)} \leftarrow$ downsample $E_0^{(i)}$ to $\lceil N_s/M \rceil$ values, else min/max of $\mathcal{D}^{(i)}$
5: $\boldsymbol{e}^{(i)}, \boldsymbol{v}^{(i)-}, \boldsymbol{v}^{(i)+}, \boldsymbol{u}^{(i)} \leftarrow \langle \hat{e}_0^{(i)} \rangle, \emptyset, \emptyset, \emptyset$ *// initialise*
6: **for** $t = 1, 2, \ldots,$ size of $\hat{\boldsymbol{e}}^{(i)} - 1$ **do** *// loop over initial bins*
7: $\boldsymbol{x}_t \leftarrow$ elements of $\boldsymbol{D}^{(i)}$ between of $\hat{e}_t^{(i)}$ and $\hat{e}_{t+1}^{(i)}$
8: $\boldsymbol{e}_{\text{new}}^{(i)}, \boldsymbol{v}_{\text{new}}^{(i)-}, \boldsymbol{v}_{\text{new}}^{(i)+}, \boldsymbol{u}_{\text{new}}^{(i)} \leftarrow \text{RefineBin1D}(\hat{e}_t^{(i)}, \hat{e}_{t+1}^{(i)}, \boldsymbol{x}_t, M, \alpha)$
9: Append $\boldsymbol{e}_{\text{new}}^{(i)}, \boldsymbol{v}_{\text{new}}^{(i)-}, \boldsymbol{v}_{\text{new}}^{(i)+}, \boldsymbol{u}_{\text{new}}^{(i)}$ to $\boldsymbol{e}^{(i)}, \boldsymbol{v}^{(i)-}, \boldsymbol{v}^{(i)+}, \boldsymbol{u}^{(i)}$
10: $\boldsymbol{c}^{(i)} \leftarrow (\boldsymbol{v}^{(i)+} + \boldsymbol{v}^{(i)-})/2$
11: $\boldsymbol{c}^{-(i)}, \boldsymbol{c}^{+(i)} \leftarrow$ bin centre bounds (Eq. 10)
12: $\boldsymbol{H}^{(i)} \leftarrow \text{Hist}(\boldsymbol{D}^{(i)}, \boldsymbol{e}^{(i)})$
13: *// 2-d histograms*
14: **for** $j = 1, 2, \ldots, i-1$ **do** *// all columns before i*
15: $\boldsymbol{e}^{(i|j)}, \boldsymbol{e}^{(j|i)} \leftarrow \boldsymbol{e}^{(i)}, \boldsymbol{e}^{(j)}$ *// initial 2-d bin edges*
16: $\boldsymbol{H}^{(ij)} \leftarrow \text{Hist}(\boldsymbol{D}^{(ij)}, \boldsymbol{e}^{(i|j)}, \boldsymbol{e}^{(j|i)})$ *// initial 2-d bin counts*
17: **for** each bin (t_i, t_j) in $\boldsymbol{H}^{(ij)}$ where $h_{t_i t_j} > M$ **do**
18: $\boldsymbol{X}_{t_i,t_j} \leftarrow$ elements of $\boldsymbol{D}^{(ij)}$ within bin (t_i, t_j)
19: $\boldsymbol{e}_{\text{new}}^{(i|j)}, \boldsymbol{e}_{\text{new}}^{(j|i)} \leftarrow \text{RefineBin2D}(e_{t_i}^{(i|j)}, e_{t_i+1}^{(i|j)}, e_{t_j}^{(j|i)}, e_{t_j+1}^{(j|i)}, \boldsymbol{X}_{t_i,t_j}, M, \alpha)$
20: Insert $\boldsymbol{e}_{\text{new}}^{(i|j)}$ into $\boldsymbol{e}^{(i|j)}$ after index t_i
21: Insert $\boldsymbol{e}_{\text{new}}^{(j|i)}$ into $\boldsymbol{e}^{(j|i)}$ after index t_j
22: $\boldsymbol{H}^{(ij)} \leftarrow \text{Hist}(\boldsymbol{D}^{(ij)}, \boldsymbol{e}^{(i|j)}, \boldsymbol{e}^{(j|i)})$ *// refined 2-d bin counts*
23: $\boldsymbol{v}^{(i|j)-}, \boldsymbol{v}^{(j|i)-}, \boldsymbol{v}^{(i|j)+}, \boldsymbol{v}^{(j|i)+} \leftarrow$ min/max values in each bin
24: $\boldsymbol{c}^{(i|j)}, \boldsymbol{c}^{(j|i)} \leftarrow (\boldsymbol{v}^{(i|j)+} + \boldsymbol{v}^{(i|j)-})/2, (\boldsymbol{v}^{(j|i)+} + \boldsymbol{v}^{(j|i)-})/2$
25: $\boldsymbol{c}^{-(i|j)}, \boldsymbol{c}^{+(i|j)}, \boldsymbol{c}^{-(j|i)}, \boldsymbol{c}^{+(j|i)} \leftarrow$ bin centre bounds (Eq. 10)
26: $\boldsymbol{u}^{(i|j)}, \boldsymbol{u}^{(j|i)} \leftarrow$ number of unique values in each bin

Algorithm 2 RefineBin1D

Inputs: bin lower edge e_L, bin upper edge e_R, vector of data values $\boldsymbol{x}$ within the bin, minimum points M, significance α

Outputs: upper bin edges $\boldsymbol{e}^*$, bin minimum values $\boldsymbol{v}^-$, bin maximum values $\boldsymbol{v}^+$, bin unique counts $\boldsymbol{u}$

1: $\boldsymbol{U} \leftarrow$ unique values in $\boldsymbol{x}$
2: $N_U \leftarrow$ number of elements in U
3: **if** $\boldsymbol{x}$ is empty **then**
4: **return** $\langle e_R \rangle, \langle e_L \rangle, \langle e_R \rangle, \langle 0 \rangle$
5: **else if** $N_U = 1$ **then**
6: **return** $\langle e_R \rangle, \langle U_0 \rangle, \langle U_0 \rangle, \langle 1 \rangle$
7: **else if** fewer than M tuples in $\boldsymbol{x}$ or IsUniform$(\boldsymbol{x}, e_L, e_R, N_U, \alpha)$ **then**
8: **return** $\langle e_R \rangle, \langle \min(\boldsymbol{U}) \rangle, \langle \max(\boldsymbol{U}) \rangle, \langle N_U \rangle$
9: **else**
10: $z \leftarrow$ select split point
11: $\boldsymbol{x}_L, \boldsymbol{x}_R \leftarrow$ split $\boldsymbol{x}$ at z
12: $\boldsymbol{e}_L^*, \boldsymbol{v}_L^-, \boldsymbol{v}_L^+, \boldsymbol{u}_L \leftarrow \text{RefineBin1D}(\boldsymbol{x}_L, e_L, z, M, \alpha)$
13: $\boldsymbol{e}_R^*, \boldsymbol{v}_R^-, \boldsymbol{v}_R^+, \boldsymbol{u}_R \leftarrow \text{RefineBin1D}(\boldsymbol{x}_R, z, e_R, M, \alpha)$
14: **return** $\langle \boldsymbol{e}_L^*, \boldsymbol{e}_R^* \rangle, \langle \boldsymbol{v}_L^-, \boldsymbol{v}_R^- \rangle, \langle \boldsymbol{v}_L^+, \boldsymbol{v}_R^+ \rangle, \langle \boldsymbol{u}_L, \boldsymbol{u}_R \rangle$

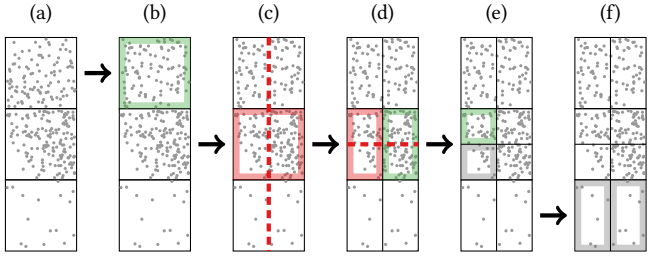

Figure 5: Illustration of two-dimensional bin refinement: (a) original data; (b) first bin is uniformly distributed (green); (c) second bin is non-uniformly distributed (red) in both dimensions, but less uniform in the vertical dimension, so a new split is added to all bins in the same column; (d) one of the resulting sub-bins is non-uniform in the vertical dimension, so a new split is added; (e) the resulting sub-bins are uniform (green) or contain fewer than M points (gray), no further splits; (f) both sub-bins from the third original bin contain fewer than M points, no further splits.

The following subsections describe PairwiseHist construction (Subsection 4.1), bin weighted centre bounds (Subsection 4.2) and PairwiseHist storage (Subsection 4.3).

4.1 Histogram construction

PairwiseHist construction is outlined in Algorithm 1, which consists of three sections: 1) extract a sample $\boldsymbol{D}$ of size N_s from dataset $\mathcal{D}$ (line 1), 2) iterate over each column to generate one-dimensional histograms (lines 3–11), and 3) iterate over each pair of columns to generate two-dimensional histograms (lines 13–26).

One-dimensional histograms are generated by first selecting initial bin edges, $\hat{\boldsymbol{e}}$, using either the bases from GreedyGD (downsampled to at most $\lceil N_s/M \rceil$) or just the min and max values of the relevant column (line 4). Each initial bin is then refined using RefineBin1D (lines 6–9), which determines whether a bin should be split or not based on a hypothesis test. The one-dimensional histograms are finalised by computing the midpoints, weighted centre bounds and bin counts (using a standard histogram function, denoted Hist) in lines 10–12.

RefineBin1D is described in detail in Algorithm 2. This is a recursive algorithm that checks whether a given bin needs to be split, i.e., if the distribution of data points within the bin is not sufficiently uniform. If so, it splits the bin and calls itself on the two newly created splits, denoted by L and R. We tested both equal-width (split at bin midpoint) and equal-depth (split at median) approaches and found equal-width to perform slightly better. Bins will not be split if they are empty (line 3), have only one unique value (line 5) or have too few data points (line 7). RefineBin1D returns the upper edges from the original bin and all new splits, as well as the bin minimum(s), maximum(s) and unique count(s).

Two-dimensional histograms are constructed using an inner loop in BuildPairwiseHist (line 14) that iterates over all columns previously iterated over by the outer loop (line 2), thereby covering all pairs of columns. For each column pair, an initial histogram is created using bin edges from the corresponding one-dimensional histograms (lines 15–16). This histogram is then refined by applying RefineBin2D to each bin with at least M tuples (lines 17–21). RefineBin2D is the two-dimensional analogue of RefineBin1D and performs a hypothesis test for uniformity on each column separately. In the case where both columns are non-uniform, the split is applied to the least uniform column. Note that any bin splits created by RefineBin2D apply only to the current column pair. That is, if a

split is applied to bin t_i in the (ij)th histogram, it does not affect any other histograms involving column i. However, the split *does* apply to all bins with the same bin index t_i in the (ij)th histogram. Fig. 5 provides an illustration of the two dimensions bin refinement process. The two-dimensional histograms are completed by calculating the bin counts, minimums, maximums, midpoints, weighted centre bounds and unique counts (lines 22–26).

Hypothesis testing in both RefineBin1D and RefineBin2D is performed using the function IsUniform. Specifically, a chi-squared test is performed against the null hypothesis that data points in the given bin are uniformly distributed between the bin edges. That is, the bin is divided into a number of sub-bins and the number of points in each sub-bin is compared to the expected number under the null hypothesis. The appropriate number of sub-bins, s, is determined using the Terrell-Scott inequality [46] as follows:

$$s = \left\lceil (2u)^{1/3} \right\rceil. \tag{2}$$

The test statistic is then:

$$\chi^2 = \sum_{r=0}^{s-1} \frac{\left(\hbar_r - \hat{\hbar}\right)^2}{\hat{\hbar}}, \tag{3}$$

where $\hat{\hbar} = h/s$ is the expected sub-bin count under the null hypothesis and $\hbar_r$ is the actual count for the rth sub-bin. The critical value, χ^2_α, is defined such that $\Pr\left(\chi^2 > \chi^2_\alpha\right) = \alpha$ and the null hypothesis is rejected (and the bin split) if $\chi^2 > \chi^2_\alpha$.

Notably, PairwiseHist construction is highly parallelisable, since each histogram and bin refinement can be computed independently, provided one-dimensional histograms are constructed first.

4.2 Bin weighted centre bounds

To improve query bounds, PairwiseHist stores weighted centres bounds for each histogram bin. The value of the weighted centre bounds depends on whether bins passed the hypothesis test or not. For bins that did not pass, the only available information on their internal data distribution is that they contain h data points, u unique values and extrema v^- and v^+. In this case, weighted centre bounds occur when $h-u+1$ points are at an extrema and one point is at each of the other unique values, which are assumed to be as close to the extrema as possible, i.e., with minimum spacing for the given data type, denoted μ. Conversely, the internal distribution of bins that pass the hypothesis test is known to be approximately uniform with respect to a given number of sub-bins. This fact can be used to derive tighter bounds, as shown in Theorem 1. Passing and non-passing bins can be distinguished by their bin count, which is at least M for passing bins and less than M for non-passing bins.

Theorem 1. *Consider a bin with count h, minimum value v^-, maximum value v^+ and u unique values. Assume this bin satisfies the hypothesis test in* IsUniform *with $s = \left\lceil (2u)^{1/3} \right\rceil$ sub-bins and critical value χ^2_α. Let $\delta = (v^+ - v^-)/s$ be the sub-bin width. The bounds for the weighted centre of the points within the bin are then*

$$c^\pm = v^- + \frac{(s \pm 1)\delta}{2} \pm \frac{\delta}{6}\sqrt{\frac{3\chi^2_\alpha(s^2-1)}{h}}. \tag{4}$$

Proof. The lower bound occurs when all points are at the lower edge of their respective sub-bins and can be expressed as follows:

$$c^- = \frac{1}{h}\sum_{r=0}^{s-1} \hbar_r (v^- + r\delta). \tag{5}$$

where $\hbar_r$ is the count for the rth sub-bin. Sub-bin counts can be expressed in terms of the expected sub-bin count plus an epsilon term, i.e., $\hbar_r = h/s + \epsilon_r$. Substituting this into Eq. 5 gives:

$$c^- = v^- + \frac{\delta(s-1)}{2} + \frac{\delta}{h}\sum_{r=0}^{s-1} r\epsilon_r. \tag{6}$$

This can be optimised using Lagrange Multipliers with constraints:

$$\sum_{r=0}^{s-1} \epsilon_r = 0 \text{ and } \chi^2_\alpha = \sum_{r=0}^{s-1} \frac{(\hbar_r - h/s)^2}{h/s} = \frac{s}{h}\sum_{r=0}^{s-1} \epsilon_r^2. \tag{7}$$

The Lagrangian is then:

$$\begin{aligned}\mathcal{L}(\epsilon_r, \lambda_1, \lambda_2) = \; & v^- + \frac{\delta(s-1)}{2} + \frac{\delta}{h}\sum_{r=0}^{s-1} r\epsilon_r \\ & + \lambda_1 \sum_{r=0}^{s-1} \epsilon_r + \lambda_2 \left(\chi^2_\alpha - \frac{s}{h}\sum_{r=0}^{s-1} \epsilon_r^2\right).\end{aligned} \tag{8}$$

Setting the partial derivative $\partial\mathcal{L}/\partial\epsilon_r$ equal to zero gives $\epsilon_r = (r\delta + h\lambda_1)/2s\lambda_2$. Substituting this into the constraints in Eq. 7 and solving for λ_1 and λ_2 gives the following expression for ϵ_r:

$$\epsilon_r = \pm\frac{2}{s}\left(r - \frac{s(s-1)}{2}\right)\sqrt{\frac{3\chi^2_\alpha h}{s^2-1}}. \tag{9}$$

Taking the negative solution and substituting this into Eq. 6 gives the desired result. A similar approach can be used for v^+. □

Thus, bin weighted centre bounds are as follows:

$$c^\pm = \begin{cases} v^\pm \mp \frac{(u-1)u\mu}{2h}, & h < M \\ v^- + \frac{(s\pm1)\delta}{2} \pm \frac{\delta}{6}\sqrt{\frac{3\chi^2_\alpha(s^2-1)}{h}}, & \text{otherwise.} \end{cases} \tag{10}$$

4.3 Storage

To minimise PairwiseHist storage requirements, we observe that bin midpoints and weighted centre bounds can easily be re-derived from other parameters and thus need not be stored. Additionally, bin counts, which require the most storage, are stored sparsely when this is more effective. For sparse encoding, we store the delta between non-zero indices and encode using Golomb coding, which is optimal for geometrically distributed data. Fig. 6 provides an overview of the storage configuration. In total, the storage requirements are:

$$S = S_{\text{params}} + S_{\text{1-d hists}} + S_{\text{2-d hists}} + S_{\text{counts}} \tag{11}$$

$$\begin{aligned} &\leq 29 + d + 4d^2 \\ &\quad + \sum_{i=1}^{d} \left(3m^{(i)} + 4\right)\left(\sum_{j=1}^{d} k^{(i|j)} - (d-1)k^{(i)}\right) \\ &\quad + \sum_{i=1}^{d}\sum_{j=1}^{d} \left\lceil k^{(i|j)} k^{(j|i)} \ell_h^{(ij)} \;/\; 8 \right\rceil \text{ bytes,} \end{aligned} \tag{12}$$

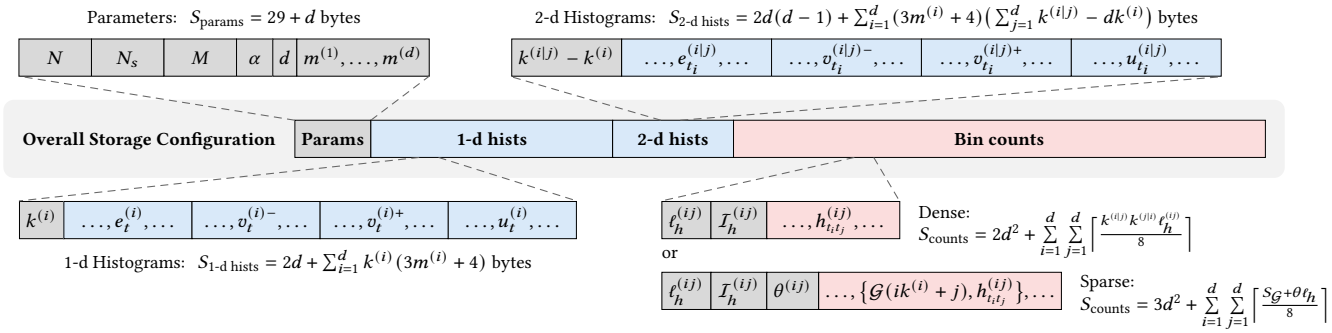

Figure 6: PairwiseHist storage configuration.

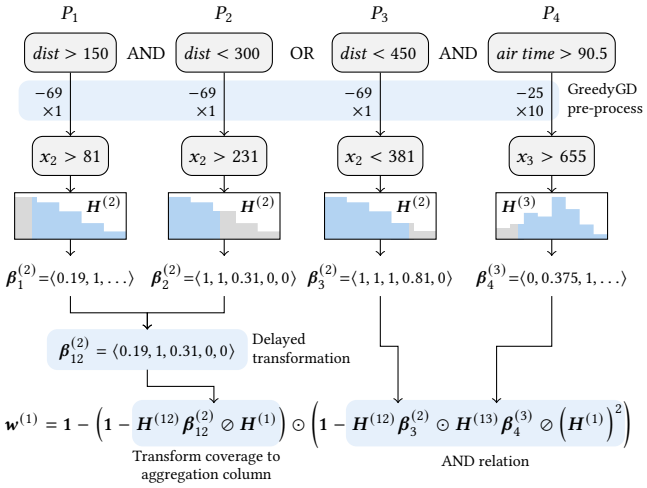

Figure 7: Partial query execution for a query aggregating on column 1 with predicates on columns 2 (dist) and 3 (air time), including applying GreedyGD pre-processing, computing coverage for each predicate and bin weightings.

where ℓ_h is the number of bits per bin count, i.e.,

$$\ell_h^{(ij)} = \left\lceil \log_2\left(1 + \max_{t_i,t_j}\left(h_{t_i t_j}^{(ij)}\right)\right)\right\rceil, \tag{13}$$

and $m^{(i)}$ is the number of bytes per value in the ith dimension. In Fig. 6, $\mathcal{I}_h^{(ij)}$ is a binary variable that indicates whether the (ij)th histogram is stored densely or sparsely, $\theta^{(ij)}$ is the number of non-zero values in $\boldsymbol{H}^{(ij)}$ and $\mathcal{G}(\cdot)$ is Golomb coding.

5 QUERY EXECUTION

This section explains how AQP tasks are executed using PairwiseHist executes (top section of Fig. 2), which is illustrated in Fig. 7.

5.1 SQL parsing

SQL queries are parsed by applying GreedyGD pre-processing to predicate literals so that they are in the same domain as the compressed data on which PairwiseHist is built. For example, in Fig. 7, the *dist* column's minimum value of 69 is subtracted from predicates 1–3, while the *air time* column's minimum value of 25 is subtracted from predicate 4. Predicate 4 is also multiplied by 10 to convert from floating point to integer.

5.2 Coverage

Given a condition P that applies to column j, we define *coverage*, $\boldsymbol{\beta}^{(j)}$, as the $k^{(j)} \times 1$ vector whose tth element is the probability that a point in the tth bin of the one-dimensional histogram for column j satisfies P. That is,

$$\beta_t^{(j)} = \Pr\left(P \,\middle|\, e_t^{(j)} \le x < e_{t+1}^{(j)}\right) \tag{14}$$

where Pr is the probability operator. This is estimated as follows for equality conditions (inverse for inequality):

$$\beta_t^{(j)} = \begin{cases} 0, & \text{condition value outside bin} \\ 1/u_t^{(j)}, & \text{otherwise,} \end{cases} \tag{15}$$

and for all other condition types (i.e., $\le$, $<$, $\ge$, $>$) as:

$$\beta_t^{(j)} = \begin{cases} 0, & v_t^{(j)-} \text{ and } v_t^{(j)+} \text{ fail } P \\ 1, & v_t^{(j)-} \text{ and } v_t^{(j)+} \text{ satisfy } P \\ 0.5, & \text{one of } v_t^{(j)-}, v_t^{(j)+} \text{ satisfy } P \;\&\; u_t^{(j)} = 2 \\ f_t(P), & \text{otherwise,} \end{cases} \tag{16}$$

where $f_t(P)$ is the fraction of the bin width (Δ_t) that satisfies P.

Coverage is estimated separately for each predicate condition, as illustrated in Fig. 7 with $\boldsymbol{\beta}_1$ to $\boldsymbol{\beta}_4$ corresponding to predicates P_1 to P_4. Groups of conditions that apply to the same column are consolidated. By 'group' we mean any group of two or more conditions that are directly connected by a single AND or OR operator. For example, in Fig. 7, $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ are consolidated into $\boldsymbol{\beta}_{12}$ since they both apply to the *dist* column and are directly connected by an AND operator. However, despite P_3 also applying to the same column, it is not consolidated since it must be first combined with P_4 due to operator precedence (AND before OR). We refer to this process as *delayed transformation* because we delay transforming coverages into weightings in the aggregation dimension (Subsection 5.3) to consolidate same-column conditions.

Coverage bounds are also computed and used to bound the query result. The source of uncertainty here is the unknown distribution of data points within partially covered bins. Applying similar logic to Subsection 4.2, we can show that, for bins that do not pass the

hypothesis test in IsUniform (i.e., bins with $h < M$), coverage bounds occur when only one data point or all but one data points satisfy P. However, for bins with at least M points, tighter bounds can be derived by considering the *partial bin count*, which is defined as the number of data points in a subset of the bin's sub-bins. Theorem 2 provides the key result for these bounds.

Theorem 2. *Consider a histogram bin with count h and u unique values. Assume that this bin satisfies the hypothesis test in* IsUniform *with $s = \lceil (2u)^{1/3} \rceil$ sub-bins and critical value χ^2_α. The minimum and maximum partial bin count over $\bar{s} \le s$ sub-bins is then:*

$$h^{\pm}_{\bar{s}|s} = \frac{h\bar{s}}{s} \pm \frac{h\bar{s}}{s}\sqrt{\frac{\chi^2_\alpha(s-\bar{s})}{h\bar{s}}}. \tag{17}$$

Proof. Use Lagrange Multipliers to optimise the partial count $\sum_{r=1}^{\bar{s}} \hbar_r$, where $\hbar_r$ is the count for the rth sub-bin, given constraints:

$$\sum_{r=1}^{s} \hbar_r = h \text{ and } \chi^2_\alpha = \sum_{r=1}^{s} \frac{(\hbar_r - h/s)^2}{h/s}. \tag{18}$$

The Lagrangian is then:

$$\mathcal{L}(\hbar_r, \lambda_1, \lambda_2) = \sum_{r=1}^{\bar{s}} \hbar_r + \lambda_1 \left(h - \sum_{r=1}^{s} \hbar_r \right) + \lambda_2 \left(\chi^2_\alpha - \sum_{r=1}^{s} \frac{(\hbar_r - h/s)^2}{h/s} \right). \tag{19}$$

Setting the partial derivative of $\mathcal{L}$ with respect to $\hbar_r$ to zero gives

$$\hbar_r = \begin{cases} \frac{h}{s} + \frac{h(1-\lambda_1)}{2s\lambda_2}, & r < \bar{s} \\ \frac{h}{s} - \frac{h\lambda_1}{2s\lambda_2}, & r \ge \bar{s} \end{cases} \tag{20}$$

Substituting this into the two constraints in Eq. 18 and solving for λ_1 and λ_2 gives the following result for $\hbar_r$:

$$\hbar_r = \begin{cases} \frac{h}{s} \pm \frac{h}{s}\sqrt{\chi^2_\alpha(s-\bar{s}) \,/\, h\bar{s}}, & r < \bar{s}, \\ \frac{h}{s} \mp \frac{h}{s}\sqrt{\chi^2_\alpha \bar{s} \,/\, h(s-\bar{s})} & r \ge \bar{s}. \end{cases} \tag{21}$$

Multiplying the solution for $r < \bar{s}$ by $\bar{s}$ gives the desired result. □

Given Theorem 2, coverage bounds $\boldsymbol{\beta}^-$ and $\boldsymbol{\beta}^+$ can be obtained by dividing the partial bin count by the total bin count with the appropriate number of sub-bins. That is,

$$\beta_t^{-(j)} = \begin{cases} \beta_t^{(j)}, & \beta_t^{(j)} \in \{0, 1\} \\ 1 \,/\, h_t^{(j)}, & \beta_t^{(j)} \notin \{0, 1\} \;\&\; h_t^{(j)} < M \\ \frac{a}{s} - \frac{a}{s}\sqrt{\chi^2_\alpha(s-a)/ha}, & \text{otherwise,} \end{cases} \tag{22}$$

$$\beta_t^{+(j)} = \begin{cases} \beta_t^{(j)}, & \beta_t^{(j)} \in \{0, 1\} \\ 1 - 1 \,/\, h_t^{(j)}, & \beta_t^{(j)} \notin \{0, 1\} \;\&\; h_t^{(j)} < M \\ \frac{b}{s} + \frac{b}{s}\sqrt{\chi^2_\alpha(s-b)/hb}, & \text{otherwise,} \end{cases} \tag{23}$$

where $s = \lceil (2u_t^{(j)})^{1/3} \rceil$ is the number of sub-bins, $a = \lfloor \beta_t^{(j)} s \rfloor$ is the number of sub-bins that are fully covered by the predicate condition and $b = \lceil \beta_t^{(j)} s \rceil$ is the number of fully or partially covered sub-bins.

5.3 Weightings

Given query predicate $\boldsymbol{P}$ containing conditions $P_1, \ldots, P_n$ and aggregation column i, we define bin *weightings*, $\boldsymbol{w}^{(i)}$, as the $k^{(i)} \times 1$ vector whose tth element is the estimated number of points in the tth bin of the one-dimensional histogram for column i that satisfy $\boldsymbol{P}$. This can be expressed as follows:

$$w_t^{(i)} = h_t^{(i)} \Pr\left(\boldsymbol{P} \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right). \tag{24}$$

For predicates that are the intersection (AND) of multiple conditions $P_1, \ldots, P_n$ and assuming conditional independence between predicate conditions, this can be expressed as follows:

$$\begin{aligned} w_t^{(i)} &= h_t^{(i)} \Pr\left(P_1 \cap P_2 \cap \ldots \cap P_n \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right) \\ &= h_t^{(i)} \prod_{\ell=1}^{n} \Pr\left(P_\ell \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right). \end{aligned} \tag{25}$$

Likewise, for predicates that are the union (OR) of multiple conditions and again assuming conditional independence between predicate conditions, weightings can be expressed as follows:

$$\begin{aligned} w_t^{(i)} &= h_t^{(i)} \Pr\left(P_1 \cup P_2 \cup \ldots \cup P_n \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right) \\ &= h_t^{(i)} \left(1 - \Pr\left(\bar{P}_1 \cap \bar{P}_2 \cap \ldots \cap \bar{P}_n \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right) \right) \\ &= h_t^{(i)} \left(1 - \prod_{\ell=1}^{n} \left(1 - \Pr\left(P_\ell \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right) \right) \right). \end{aligned} \tag{26}$$

We then observe that

$$\Pr\left(P_\ell \,\middle|\, e_t^{(i)} \le x < e_{t+1}^{(i)} \right) = \frac{1}{h_t^{(i)}} \left[\boldsymbol{H}^{(ij)} \boldsymbol{\beta}^{(j)} \right]_t, \tag{27}$$

which allows us to express weightings as follows:

$$\boldsymbol{w}^{(i)} = \begin{cases} \prod_{\ell=1}^{n} \boldsymbol{H}^{(ij)} \boldsymbol{\beta}^{(j)} \oslash \boldsymbol{H}^{(i)}, & \text{intersection} \\ 1 - \prod_{\ell=1}^{n} \left(1 - \boldsymbol{H}^{(ij)} \boldsymbol{\beta}^{(j)} \oslash \boldsymbol{H}^{(i)} \right), & \text{union} \end{cases} \tag{28}$$

where the product (Π) is element-wise (Hadamard). By combining these results, arbitrary combinations of AND and OR relations can be processed. Note that due to assuming conditional independence, Eq. 28 may not be reliable for highly correlated data. This is especially true for multiple conditions on the same column, which are clearly not (conditionally) independent, and thus why we perform delayed transformation to consolidate such conditions prior to computing bin weightings.

Weightings bounds, $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$, are computed using Eq. 28, using low and high coverage bounds, respectively. If PairwiseHist is constructed from a data sample, these bounds are widened to account for additional uncertainty due to sampling. That is, the bounds are replaced by their outer two-sided 98-percentile confidence interval bounds. Variance is estimated according to the Binomial distribution as $\beta_t(1-\beta_t)$ where $\beta_t = w_t/h_t$. Including compensation for finite population size, the weightings bounds are updated as follows:

$$w_t^{\pm} \leftarrow w_t^{\pm} \pm z_{0.98}\sqrt{\beta_t^{\pm}\left(1-\beta_t^{\pm}\right)\frac{N-N_s}{N-1}}, \tag{29}$$

where $z_{0.98}$ is the value of the standard normal density function corresponding to a two-sided 98-percentile confidence interval.

Table 3: Aggregation functions

Aggregation	Estimate	Lower bound	Upper bound
COUNT	$\|\boldsymbol{w}\|_1 \,/\, \rho$	$\|\boldsymbol{w}^-\|_1 \,/\, \rho$	$\|\boldsymbol{w}^+\|_1 \,/\, \rho$
SUM	$\boldsymbol{w} \cdot \boldsymbol{c} \,/\, \rho$	$\boldsymbol{w}^- \cdot \boldsymbol{c}^- \,/\, \rho$	$\boldsymbol{w}^+ \cdot \boldsymbol{c}^+ \,/\, \rho$
AVG	$\boldsymbol{w} \cdot \boldsymbol{c} \,/\, \|\boldsymbol{w}\|_1$	$\min_{\{\boldsymbol{w}^-,\, \boldsymbol{w}^+\}} \{\boldsymbol{w}^\bullet \cdot \boldsymbol{c}^- \,/\, \|\boldsymbol{w}^\bullet\|_1\}$	$\max_{\{\boldsymbol{w}^-,\, \boldsymbol{w}^+\}} \{\boldsymbol{w}^\bullet \cdot \boldsymbol{c}^+ \,/\, \|\boldsymbol{w}^\bullet\|_1\}$
MIN	$\begin{cases} v^+_{t^*}, & \text{single-column, } u_{t^*} = 2 \;\&\; w_{t^*} < \frac{h_{t^*}}{2} \\ v^-_{t^*}, & \text{otherwise.} \end{cases}$	$\begin{cases} v^+_{t^*}, & \text{single-column, } u_{t^*} = 2 \;\&\; w^+_{t^*} < \frac{h_{t^*}}{5} \\ v^-_{t^*}, & \text{otherwise.} \end{cases}$	$\begin{cases} v^+_{t^*} - a\delta_{t^*}, & \text{single-column, } u_{t^*} > 2 \;\&\; h_{t^*} > M \\ v^+_{t^*}, & \text{otherwise,} \end{cases}$
MAX	$\begin{cases} v^-_{t^*}, & \text{single-column, } u_{t^*} = 2 \;\&\; w_{t^*} < \frac{h_{t^*}}{2} \\ v^+_{t^*}, & \text{otherwise.} \end{cases}$	$\begin{cases} v^-_{t^*} + a\delta_{t^*}, & \text{single-column, } u_{t^*} > 2 \;\&\; h_{t^*} > M \\ v^-_{t^*}, & \text{otherwise,} \end{cases}$	$\begin{cases} v^-_{t^*}, & \text{single-column, } u_{t^*} = 2 \;\&\; w^+_{t^*} < \frac{h_{t^*}}{5} \\ v^+_{t^*}, & \text{otherwise.} \end{cases}$
MEDIAN	$\begin{cases} v^-_{t^*}, & u_{t^*} = 2 \;\&\; f_{t^*} < 0.5 \\ v^+_{t^*}, & u_{t^*} = 2 \;\&\; f_{t^*} \geq 0.5 \\ v^-_{t^*} + \Delta_{t^*} f_{t^*}, & \text{otherwise,} \end{cases}$	$v^-_{t^*}$	$v^+_{t^*}$
VAR	$\boldsymbol{w} \cdot \boldsymbol{c}^2 \,/\, \|\boldsymbol{w}\|_1 - (\boldsymbol{w} \cdot \boldsymbol{c} \,/\, \|\boldsymbol{w}\|_1)^2$.	$\min_{\{\boldsymbol{w}^-,\, \boldsymbol{w}^+\}} \left\{ \begin{array}{l} \boldsymbol{w}^\bullet \cdot (\xi^-)^2 \,/\, \|\boldsymbol{w}^\bullet\|_1 \\ \quad - (\boldsymbol{w}^\bullet \cdot \xi^- \,/\, \|\boldsymbol{w}^\bullet\|_1)^2 \end{array} \right\}$	$\max_{\{\boldsymbol{w}^-,\, \boldsymbol{w}^+\}} \left\{ \begin{array}{l} \boldsymbol{w}^\bullet \cdot (\xi^+)^2 \,/\, \|\boldsymbol{w}^\bullet\|_1 \\ \quad - \left(\boldsymbol{w}^\bullet \cdot \xi^+ \,/\, \|\boldsymbol{w}^\bullet\|_1\right)^2 \end{array} \right\}$

5.4 Aggregation

This section describes the mathematical formulations for query aggregation, which are listed in Table 3, along with corresponding bounds. Currently, seven aggregation functions are supported, namely COUNT, SUM, AVG, MIN, MAX, MEDIAN and VAR.

5.4.1 COUNT. This can be estimated by summing the weightings vector and then dividing by the sampling ratio, ρ.

5.4.2 SUM. This can be estimated as the weighted sum of bin midpoints, $\boldsymbol{c}$, using the estimated bin weightings, $\boldsymbol{w}$. As with COUNT, the result must also be scaled by the sampling ratio, ρ.

5.4.3 AVG. This can be estimated as the weighted mean of bin midpoints, $\boldsymbol{c}$, using the estimated bin weightings, $\boldsymbol{w}$. The bounds follow the same formulation, but naturally use the appropriate bounds for bin midpoints. Additionally, each bound is estimated using both bin weightings extrema and either the minimum or maximum is returned. For example, the lower bound is the minimum over $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$ where $\boldsymbol{w}^\bullet$ is a placeholder variable.

5.4.4 MIN. Estimating MIN requires identifying the index, t^*, of the first bin with non-zero weighting, i.e.,

$$t^* = \min_t \{t, \text{ s.t. } w_t > 0\}. \tag{30}$$

In most cases, the estimate is the minimum value for this bin, $v^-_{t^*}$. For the special case where the query involves only a single column (for aggregation and all predicates), there are only two unique values in the t^*th bin (i.e., $u_{t^*} = 2$) and the bin coverage is less than half (i.e., $w_{t^*} < h_{t^*}/2$), then the bin maximum, $v^+_{t^*}$, is a better estimate.

The lower bound follows the same formulation, except that t^* must be determined from the bin weightings upper bound (to include the maximum range), i.e.,

$$t^* = \min_t \{t, \text{ s.t. } w^+_t > 0\} \qquad \text{(lower bound)}. \tag{31}$$

Conversely, the bin corresponding to the upper bound is identified using the bin weightings lower bound. A higher threshold is used to ensure a higher likelihood of non-zero true bin coverage, i.e.,

$$t^* = \min_t \{t, \text{ s.t. } w^-_t > 1/2\} \qquad \text{(upper bound)}. \tag{32}$$

A tighter upper bound can be obtained for queries involving a single column in certain cases by considering the sub-bins. That is, if the relevant bin passed the uniformity hypothesis test, and hence $h \geq M$, then we may assume that data is uniformly distributed across the sub-bins. Thus, we can compute the number of sub-bins that are fully covered as $a = \lfloor s w^-_{t^*} / h_{t^*} \rfloor$ and reduce the upper bound by this number of sub-bin widths, δ_{t^*}.

5.4.5 MAX. This is the inverse of MIN. That is, we identify the index, t^*, of the *last* bin with non-zero weighting, i.e.,

$$t^* = \max_t \{t, \text{ s.t. } w_t > 0\}, \tag{33}$$

and use this to determine the estimates. Similar, but inverse, formulations apply for the lower and upper bounds.

5.4.6 MEDIAN. This is estimated by first identifying the index, t^*, of the bin that contains the median. That is,

$$t^* = \min_t \left\{ t, \text{ s.t. } \sum_{\tau=0}^{t} w_\tau \geq \frac{1}{2} \|\boldsymbol{w}\|_1 \right\}. \tag{34}$$

The estimate is then the bin minimum, $v^-_{t^*}$ plus the bin width, $\Delta_{t^*} = v^+_{t^*} - v^-_{t^*}$, multiplied by the fraction of the bin weighting that is below the median, f_{t^*}, which is calculated as follows:

$$f_{t^*} = \frac{1}{w_{t^*}} \left(\frac{1}{2} \|\boldsymbol{w}\|_1 - \sum_{t=0}^{t^*-1} w_t \right) \tag{35}$$

If the bin contains only two unique values (i.e., $u_{t^*} = 2$), the bin minimum or maximum is returned instead, depending on the value of f_{t^*}.

MEDIAN bounds require identifying the minimum and maximum bin indices that could correspond to the median, i.e.,

$$t^* = \min_{\{\boldsymbol{w}^-,\ \boldsymbol{w}^+\}} \left\{ \min_t \left\{ t,\ \text{s.t.} \sum_{\tau=0}^{t} w_\tau^\bullet \geq \frac{1}{2} \right\} \right\} \quad \text{(lower bound)} \quad (36)$$

$$t^* = \max_{\{\boldsymbol{w}^-,\ \boldsymbol{w}^+\}} \left\{ \min_t \left\{ t,\ \text{s.t.} \sum_{\tau=0}^{t} w_\tau^\bullet \geq \frac{1}{2} \right\} \right\} \quad \text{(upper bound)} \quad (37)$$

5.4.7 VAR. This is estimated as the difference between the weighted mean square value and the square of the estimated mean. The idea for the bounds is to assume that all points within a bin are either as far from the mean as possible (upper bound) or as close to the mean as possible (lower bound). This is expressed using ξ^- and ξ^+, which represent the (assumed) location of points within each bin for the purpose of estimating the bounds and are defined as follows:

$$\xi_t^- = \begin{cases} v_t^+, & v_t^+ < AVG \\ v_t^-, & v_t^- > AVG \\ AVG, & \text{otherwise,} \end{cases} \quad (38)$$

$$\xi_t^+ = \begin{cases} v_t^-, & |AVG - v_t^-| > |v_t^+ - AVG| \\ v_t^+, & \text{otherwise.} \end{cases} \quad (39)$$

where AVG is the estimated mean. Thus, for the lower bound, bins are represented by whichever of the bin minimum or maximum is closest to the estimated mean, while the bin that straddles the mean is represented by the mean itself. Conversely, for the upper bound, bins are represented by whichever of the bin minimum or maximum is furthest from the mean. As with AVG and MEDIAN queries, the bounds are evaluated for both weightings extrema and the appropriate min/max is returned.

6 PERFORMANCE EVALUATION

To evaluate PairwiseHist, 11 real-world datasets were used, which are summarised in Table 4. The *Basement*, *Current* and *Furnace* datasets [36, 37] contain electrical meter data for different areas of a house. *Gas* [21], *Light* [44], *Power* [17] and *Temp* [43] contain multifaceted IoT sensor data from a single source, while *Aqua* [52] and *Build* [20] contain IoT sensor data combined from multiple sources (aquaponics ponds and building rooms, respectively) with several data columns each and a shared timestamp column. *Aqua* and *Build* thus contain many null values due to asynchronous sampling. *Flights* [53] and *Taxi* [9] contain records of individual trips and include several categorical fields, as well as missing values. *Flights* is commonly used in AQP literature (e.g. [12, 15, 19, 28, 34, 49]), while *Power* is used occasionally (e.g. [61, 62]). Overall, these datasets encompass a wide variety of data types, dimensionality, sizes and both sensor and non-sensor data.

We implemented PairwiseHist in Python 3.11 and compared it to state-of-the-art AQP techniques DeepDB [19] and DBEst++ [34] using their corresponding Python implementations [18, 33]. These techniques were chosen for comparison due to their leading performance (see Table 1) and code availability.

Our evaluation is divided into two parts: initial experiments on the original datasets and comprehensive experiments on scaled-up versions of the *Power* and *Flights* datasets. For the initial experiments, we randomly generated 100 single-predicate queries for each dataset with aggregation functions COUNT, SUM and AVG and minimum selectivity of 10^{-5}. For the scaled-up experiments, IDEBench [12] was used to scale the datasets up to one billion rows, resulting in sizes of 40 GB and 130 GB, respectively. We then randomly generated 445 and 427 test queries for *Power* and *Flights*, respectively, including all seven aggregation functions supported by PairwiseHist, 1–5 predicate conditions and minimum selectivity of 10^{-6}. Due to aforementioned limitations of DeepDB and DBEst++ (Section 2), DeepDB supports only 146 queries (80 for *Power* and 66 for *Flights*), while DBEst++ supports just 86 queries (41 for *Power* and 45 for *Flights*). Also, since DBEst++ requires many models to support different query templates (see Section 2), we include all DBEst++ models required to support the same queries as PairwiseHist when comparing synopsis size.

Our experimental setup consisted of an Intel(R) Xeon(R) Gold 6130 2.10 GHz CPU with 24 GB RAM available during synopsis construction and 6 GB during query execution. Default parameters were used for DeepDB and DBEst++, as well as GreedyGD. All experiments were performed with M set to 1% of N_s (e.g. $M = 10^3$ for $N_s = 10^5$) and α set to 0.001.

Table 4: Datasets used for evaluation

Dataset		Rows	Cols.	Size (MB)
Aqua	Aquaponics sensors [52]	913 465	13	66.7
Basement	Basement power [36, 37]	1 051 200	12	50.5
Build	Smart building systems [20]	14 381 639	7	402.7
Current	Electric meters current [36, 37]	1 051 200	24	100.9
Flights*	Flight delays & cancellations [53]	5 819 079	32	756.5
Furnace	Furnace power [36, 37]	1 051 200	12	50.5
Gas	Home gas sensor [21]	928 991	12	44.6
Light	IoT light detection [44]	405 184	9	19.9
Power	Home power consumption [17]	2 049 280	10	82.0
Taxis	Chicago taxi trips 2020 [9]	3 889 032	23	1 753.9
Temp	Temperature sensor [43]	10 553 597	5	369.4

*Other works typically use only 12 columns (e.g. [19]), however, we use all 32 columns.

6.1 Initial experiments

The median query error and synopsis size for each dataset is shown in Fig. 8 for PairwiseHist, DeepDB and DBEst++ with 100k and 10k samples. As can be seen in Fig. 8(a), PairwiseHist has the lowest error on 10 out of 11 datasets. Indeed, even with a mere 10k samples, PairwiseHist outperforms DeepDB with 100k samples on 6 out of 11 datasets. Overall, for 100k samples, PairwiseHist has a median error of just 0.28%, compared to 0.73% for DeepDB and 28.9% for DBEst++. In terms of synopsis size, PairwiseHist is typically 1–2 orders of magnitude smaller. For 100k samples, the mean size for PairwiseHist is just 0.48 MB, compared to 11.5 MB for DeepDB and 36.3 MB for DBEst++. That is, PairwiseHist is at least 2.6× as accurate while requiring 24× less storage.

6.2 Parameter sensitivity

In general, higher N_s, higher α and lower M all correspond to higher accuracy at the cost of larger synopsis size and longer construction

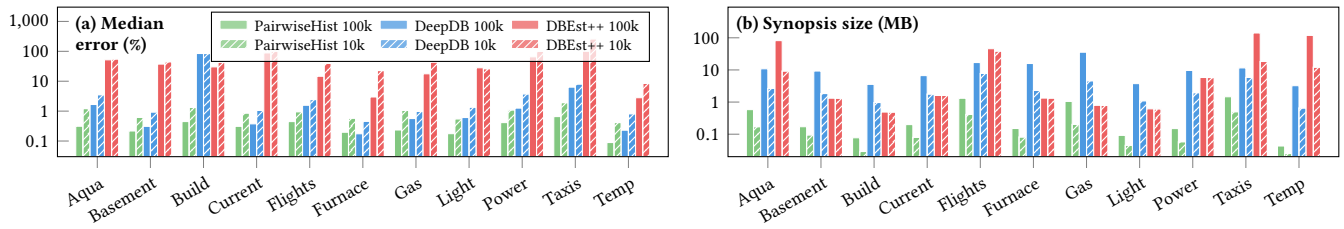

Figure 8: Error performance and storage requirements across 11 real-world datasets.

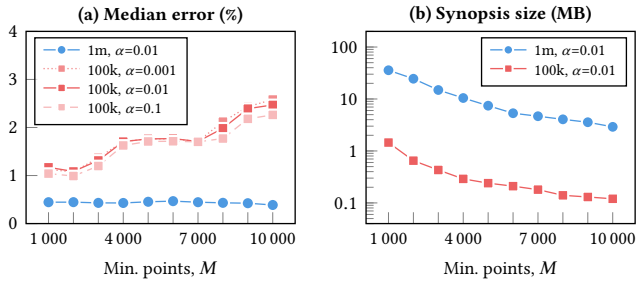

Figure 9: PairwiseHist performance on the scaled-up *Flights* dataset for different parameter sets.

Table 5: Median relative error (%).

	Power dataset			*Flights* dataset		
Aggregation	PH	DeepDB	DBEst++	PH	DeepDB	DBEst++
COUNT	0.19	**0.05**	24.82	**0.38**	0.41	21.65
SUM	**0.32**	14.18	56.46	**1.15**	1.72	3.55
AVG	**0.42**	0.50	17.86	0.39	**0.28**	16.95
VAR	**0.84**	-	98.50	**1.67**	-	100.00
MIN	**0.00**	-	-	**0.00**	-	-
MAX	**1.25**	-	-	**4.41**	-	-
MEDIAN	**0.00**	-	-	**0.29**	-	-
Overall	**0.20**	0.45	56.46	**0.43**	0.64	28.42

PH = PairwiseHist: 1 million samples, DeepDB: 1 million samples, DBEst++: 100k samples.

time. However, N_s has greatest impact on performance, while α has minimal impact. This is illustrated in Fig. 9 for the scaled-up *Flights* dataset, where α has near-zero impact except on accuracy for N_s = 100k in some cases. In our tests, we have also observed that construction time scales linearly with N_s and query latency is largely consistent across different parameter sets.

6.3 Accuracy

Query Accuracy. Table 5 presents the median error by dataset and aggregation function for the scaled-up experiments. As can be seen, PairwiseHist (denoted PH) performs well across all aggregation functions and delivers between 1.5–2.3× better overall accuracy than DeepDB with median errors of just 0.20% and 0.43% compared to 0.45% and 0.64% for the two datasets. Note that a smaller sample size was used for DBEst++ were due to its prohibitively long training time (see Subsection 6.6).

Table 6: Bounds accuracy rate and width.

	Correct rate (%)		Width (%)	
Dataset	PairwiseHist	DeepDB	PairwiseHist	DeepDB
Power (original)	**70.0**	40.0	4.4	**0.7**
Power (1 billion)	**80.0**	51.2	3.4	**0.6**
Flights (original)	**78.8**	50.0	8.7	**3.0**
Flights (1 billion)	**78.8**	75.8	4.3	**2.3**

We also compared the distribution of query errors over the subset of queries supported by DeepDB (Fig. 10(a)) and DBEst++ (Fig. 10(b)) as CDF plots, while the error distribution for PairwiseHist over all queries is shown in (Fig. 10(c)). As can be seen, PairwiseHist provides a better error distribution in each case. Indeed, with just 100k samples, PairwiseHist achieves a higher probability of sub-1% error than DeepDB with 1 million samples. Overall, 85.1% of queries have sub-10% error with PairwiseHist, as highlighted in Fig. 10(c).

During our evaluation, DeepDB was observed to perform significantly worse on real-world data compared to IDEBench-generated synthetic data. To demonstrate this, we generated synthetic versions of the *Power* and *Flights* datasets using IDEBench with the same number of rows as the original data and tested identical queries on them using DeepDB and PairwiseHist. The resulting median errors are shown in Fig. 10(d). As can be seen, DeepDB performs far worse on the real data compared to the IDEBench-generated data. IDEBench generates synthetic data by applying normalisation and Gaussian models. This suggests that, while DeepDB performs well on standard test data, it may not perform as effectively in the real world, where data is less well-behaved. PairwiseHist, on the other hand, performs consistently well and has up to 31× lower error than DeepDB on the real datasets.

Query Bounds. Query bounds should be both accurate (i.e., contain the true result) and narrow. Table 6 lists the percentage of queries for which DeepDB and PairwiseHist provide accurate bounds and the median bound widths as a percentage of the exact result for the subset of queries supported by DeepDB (DBEst++ does not provide bounds). A significance value of 0.99 was used for DeepDB. As shown, PairwiseHist provides more accurate bounds than DeepDB, especially for the real-world datasets. DeepDB has consistently narrower bounds, but given their lower accuracy, this may indicate that its bounds are overly optimistic.

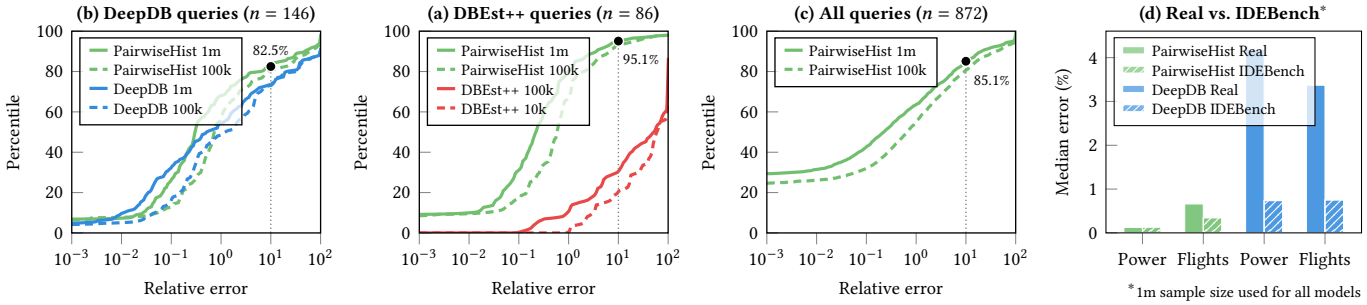

Figure 10: (a)–(c) CDF plots of query error for different subsets of queries across both datasets, (d) query performance on the original real-world data and IDEBench-generated data of the same size.

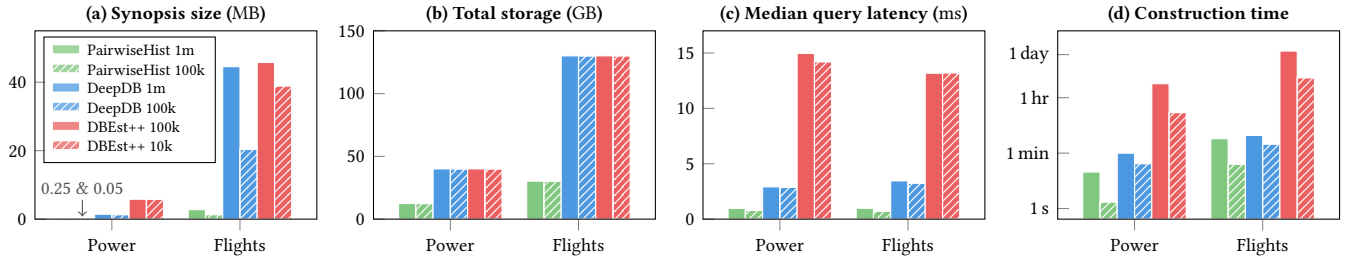

Figure 11: Storage and runtime performance comparison on the scaled-up datasets.

6.4 Storage requirements

Synopsis sizes are shown in Fig. 11(a). As can be seen, PairwiseHist requires the least storage in all cases and is at least 11× smaller than DeepDB and DBEst++ (0.25 MB vs. 2.75 MB for *Power* with 1m samples). Additionally, with PairwiseHist built directly on compressed data, total storage requirements are reduced, since data can be permanently stored in compressed format. Total storage requirements are shown in Fig. 11(b), where PairwiseHist delivers savings of 3.2–4.3×.

6.5 Query latency

Median query latency, including the time to calculate query bounds (for PairwiseHist and DeepDB), is shown in Fig. 11(c). As can be seen, PairwiseHist is significantly faster than the state-of-the-art approaches with an overall median latency of just 0.94 ms (for 1m samples), which is 3.5× faster than DeepDB and 15× faster than DBEst++. PairwiseHist's low latency can be attributed to most aggregations requiring just a handful of small matrix multiplications. The corresponding median latency for exact query processing using SQLite, which we used for the ground truth, was 306.8 seconds, which makes PairwiseHist over 300,000× faster.

6.6 Construction time

Finally, synopsis construction times are displayed in Fig. 11(d). As can be seen, PairwiseHist consistently requires the least time, being 1.2–4× faster than DeepDB, while DBEst++ is more than two orders of magnitude slower. Indeed, for the Flights dataset, DBEst++ requires over 30 hours for 100k samples, while PairwiseHist requires less than 3 minutes with 1 million samples.

7 CONCLUSION

In this paper, we propose a novel AQP technique called PairwiseHist and a novel framework for AQP that leverages data compression to reduce overall storage requirements. By using a collection of histograms approach, efficient storage encoding and various query execution optimisations, we are able to simultaneously deliver significant improvements in terms of accuracy, latency, synopsis size and construction time compared to state-of-the-art AQP methods. In future work, we intend to investigate histogram updates, online refinement and multi-table support.

ACKNOWLEDGMENTS

This work is supported by the Analytics Straight on Compressed IoT Data (Light-IoT) project (Grant No. 0136-00376B), granted by the Danish Council for Independent Research, and Aarhus University's DIGIT Centre. Computation was partially performed on the UCloud system, which is managed by the eScience Center at the University of Southern Denmark.

REFERENCES

[1] Jayadev Acharya, Ilias Diakonikolas, Chinmay Hegde, Jerry Zheng Li, and Ludwig Schmidt. 2015. Fast and Near-Optimal Algorithms for Approximating Distributions by Histograms. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.* https://doi.org/10.1145/2745754.2745772

[2] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. 2014. Knowing when you're

wrong: Building Fast and Reliable Approximate Query Processing Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 481–492. https://doi.org/10.1145/2588555.2593667

[3] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB. In *Proceedings of the 8th ACM European Conference on Computer Systems*. https://doi.org/10.1145/2465351.2465355

[4] Hossein Ahmadvand, Maziar Goudarzi, and Fouzhan Foroutan. 2019. Gapprox: using Gallup approach for approximation in Big Data processing. *Journal of Big Data* 6, 1 (2019). https://doi.org/10.1186/s40537-019-0185-4

[5] Pritom Saha Akash, Wei-Cheng Lai, and Po-Wen Lin. 2022. Online Aggregation based Approximate Query Processing: A Literature Survey. https://doi.org/10.48550/ARXIV.2204.07125

[6] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic Sample Selection for Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 539–550. https://doi.org/10.1145/872757.872822

[7] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/375663.375686

[8] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *ACM International Conference on Management of Data (SIGMOD/PODS'17)*. ACM. https://doi.org/10.1145/3035918.3056097

[9] City of Chicago. 2022. Taxi Trips - 2020. https://data.cityofchicago.org/Transportation/Taxi-Trips-2020/r2u4-wwk3 Accessed: 18 Feb, 2024.

[10] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2011. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches, In Foundations and Trends in Databases. *Foundations and Trends in Databases* 4, 1-3, 1–294. https://doi.org/10.1561/1900000004

[11] Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. 2018. Fast and Sample Near-Optimal Algorithms for Learning Multidimensional Histograms. In *Proceedings of the 31st Conference On Learning Theory*, Vol. 75. 819–842.

[12] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. IDEBench: A Benchmark for Interactive Data Exploration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM. https://doi.org/10.1145/3318464.3380574

[13] Mohammadali Fallahian, Mohsen Dorodchi, and Kyle Kreth. 2022. GAN-based Tabular Data Generator for Constructing Synopsis in Approximate Query Processing: Challenges and Solutions. https://doi.org/10.48550/ARXIV.2212.09015

[14] Marcell Fehér, Daniel E. Lucani, and Ioannis Chatzigeorgiou. 2022. An Adaptive Column Compression Family for Self-Driving Databases. https://doi.org/10.48550/arXiv.2209.02334 arXiv:2209.02334v1.

[15] Shaddy Garg, Subrata Mitra, Tong Yu, Yash Gadhia, and Arjun Kashettiwar. 2023. Reinforced Approximate Exploratory Data Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 6 (2023), 7660–7669. https://doi.org/10.1609/aaai.v37i6.25929

[16] Christian Göttel, Lars Nielsen, Niloofar Yazdani, Pascal Felber, Daniel E. Lucani, and Valerio Schiavoni. 2020. Hermes: enabling energy-efficient IoT networks with generalized deduplication. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. ACM. https://doi.org/10.1145/3401025.3404098

[17] Georges Hebrail and Alice Berard. 2012. Individual household electric power consumption Data Set. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption Accessed: 18 Feb, 2024.

[18] Benjamin Hilprecht. 2020. deepdb-public. https://github.com/DataManagementLab/deepdb-public GitHub repository. Accessed: 1 Apr, 2023..

[19] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries, In Proceedings of the VLDB Endowment. *Proceedings of the VLDB Endowment* 13, 7, 992–1005. https://doi.org/10.14778/3384345.3384349

[20] Dezhi Hong, Quanquan Gu, and Kamin Whitehouse. 2017. High-dimensional Time Series Clustering via Cross-Predictability. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Vol. 54. 642–651. https://www.kaggle.com/datasets/ranakrc/smart-building-system Accessed: 18 Feb, 2024.

[21] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai F Rulkov, and Irene Rodriguez-Lujan. 2016. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems* 157 (2016), 169–176. https://doi.org/10.1016/j.chemolab.2016.07.004

[22] Aaron Hurst, Daniel E. Lucani, Ira Assent, and Qi Zhang. 2022. GLEAN: Generalized Deduplication Enabled Approximate Edge Analytics. *IEEE Internet of Things Journal* (2022). https://doi.org/10.1109/JIOT.2022.3166455

[23] Aaron Hurst, Daniel E. Lucani, and Qi Zhang. 2024. GreedyGD: Enhanced Generalized Deduplication for Direct Analytics in IoT. *IEEE Transactions on Industrial Informatics* (2024), 1–9. https://doi.org/10.1109/tii.2024.3353913

[24] Aaron Hurst, Qi Zhang, Daniel E. Lucani, and Ira Assent. 2021. Direct Analytics of Generalized Deduplication Compressed IoT Data. In *IEEE Global Communications Conference (GLOBECOM)*. https://doi.org/10.1109/GLOBECOM46510.2021.9685589

[25] Yannis E. Ioannidis and Viswanath Poosala. 1995. Balancing histogram optimality and practicality for query result size estimation. *ACM SIGMOD Record* 24, 2 (1995), 233–244. https://doi.org/10.1145/568271.223841

[26] Linghe Kong, Jinlin Tan, Junqin Huang, Guihai Chen, Shuaitian Wang, Xi Jin, Peng Zeng, Muhammad Khan, and Sajal K. Das. 2022. Edge-computing-driven Internet of Things: A Survey. *Comput. Surveys* 55, 8 (2022), 1–41. https://doi.org/10.1145/3555308

[27] Moritz Kulessa, Benjamin Hilprecht, Alejandro Molina, Knowledge Engineering, Group Data, Management Lab, Machine Learning, and Lab. 2019. Towards Model-based Approximate Query Processing. In *1st International Workshop on Applied AI for Database Systems and Applications*.

[28] Moritz Kulessa, Alejandro Molina, Carsten Binnig, Benjamin Hilprecht, and Kristian Kersting. 2018. Model-based Approximate Query Processing. https://doi.org/10.48550/ARXIV.1811.06224

[29] Taewhi Lee, Kihyuk Nam, Choon Seo Park, and Sung-Soo Kim. 2022. Exploiting Machine Learning Models for Approximate Query Processing. In *IEEE International Conference on Big Data*. https://doi.org/10.1109/bigdata55660.2022.10020252

[30] Kaiyu Li and Guoliang Li. 2018. Approximate Query Processing: What is New and Where to Go? *Data Science and Engineering* 3, 4 (2018), 379–397. https://doi.org/10.1007/s41019-018-0074-4

[31] Kaiyu Li, Yong Zhang, Guoliang Li, Wenbo Tao, and Ying Yan. 2019. Bounded Approximate Query Processing. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2019), 2262–2276. https://doi.org/10.1109/tkde.2018.2877362

[32] Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. 2021. Combining Aggregation and Sampling (Nearly) Optimally for Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/3448016.3457277

[33] Qingzhi Ma. 2022. DBEstClient. https://github.com/qingzma/DBEstClient GitHub repository. Accessed: 1 Apr, 2023..

[34] Qingzhi Ma, Ali M. Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned Approximate Query Processing: Make it Light, Accurate and Fast. In *Conference on Innovative Data Systems Research*.

[35] Qingzhi Ma and Peter Triantafillou. 2019. DBEst: Revisiting Approximate Query Processing Engines with Machine Learning Models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1553–1570. https://doi.org/10.1145/3299869.3324958

[36] Stephen Makonin. 2016. AMPds2: The Almanac of Minutely Power dataset (Version 2). https://doi.org/10.7910/DVN/FIE0S4

[37] Stephen Makonin, Bradley Ellert, Ivan V. Bajić, and Fred Popowich. 2016. Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014. *Scientific Data* 3 (2016). https://doi.org/10.1038/sdata.2016.37

[38] Magnus Müller, Guido Moerkotte, and Oliver Kolb. 2018. Improved selectivity estimation by combining knowledge from sampling and synopses, In Proceedings of the VLDB Endowment. *Proceedings of the VLDB Endowment* 11, 9, 1016–1028. https://doi.org/10.14778/3213880.3213882

[39] Sabuzima Nayak, Ripon Patgiri, Lilapati Waikhom, and Arif Ahmed. 2022. A review on Edge analytics: Issues, challenges, opportunities, promises, future directions, and applications. *Digital Communications and Networks* (2022). https://doi.org/10.1016/j.dcan.2022.10.016 arXiv:2107.06835

[40] Lars Nielsen, Rasmus Vestergaard, Niloofar Yazdani, Prasad Talasila, Daniel E. Lucani, and Marton Sipos. 2019. Alexandria: A Proof-of-Concept Implementation and Evaluation of Generalised Data Deduplication. In *IEEE (GLOBECOM) Workshops*. https://doi.org/10.1109/gcwkshps45667.2019.9024368

[41] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/3183713.3196905

[42] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. 2018. AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1477–1492. https://doi.org/10.1145/3183713.3183747

[43] Matthew Porter. 2021. Temperature IoT on GCP. https://www.kaggle.com/datasets/mattpo/temperature-iot-on-gcp Accessed: 18 Feb, 2024.

[44] Aashna Prasad. 2020. ML prediction for Light Detection Sensor IoT. https://www.kaggle.com/datasets/aashnaprasad/ml-prediction-for-lightdetection-sensor-iot Accessed: 18 Feb, 2024.

[45] Viktor Sanca, Periklis Chrysogelos, and Anastasia Ailamaki. 2023. LAQy: Efficient and Reusable Query Approximations via Lazy Sampling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[46] David W. Scott. 2009. Sturges' rule. *WIREs Computational Statistics* 1, 3 (2009), 303–306. https://doi.org/10.1002/wics.35

[47] Hadi Sehat, Anders Lindskov Kloborg, Christian Mørup, Elena Pagnin, and Daniel E. Lucani. 2022. Bonsai: A Generalized Look at Dual Deduplication.

[48] Michael Shekelyan, Anton Dignös, and Johann Gamper. 2017. DigitHist: A Histogram-Based Data Summary with Tight Error Bounds. In *Proceedings of the VLDB Endowment*, Vol. 10. 1514–1525. https://doi.org/10.14778/3137628.3137658

[49] Nikhil Sheoran, Subrata Mitra, Vibhor Porwal, Siddharth Ghetia, Jatin Varshney, Tung Mai, Anup Rao, and Vikas Maddukuri. 2022. Conditional Generative Model Based Predicate-Aware Query Approximation. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 8 (2022), 8259–8266. https://doi.org/10.1609/aaai.v36i8.20800

[50] Samriddhi Singla and Ahmed Eldawy. 2022. *Flexible Computation of Multi-dimensional Histograms*. New York, NY, USA, Chapter 13, 119–130. https://doi.org/10.1145/3548732.3548746

[51] Hien To, Kuorong Chiang, and Cyrus Shahabi. 2013. Entropy-based histograms for selectivity estimation. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management (CIKM)*. ACM Press. https://doi.org/10.1145/2505515.2505756

[52] Udanor Collins, Blessing Ogbuokiri, and Nweke Onyinye. 2022. Sensor Based Aquaponics Fish Pond Datasets. https://doi.org/10.34740/kaggle/dsv/3748790

[53] USA Department of Transportation. 2016. 2015 Flight Delays and Cancellations. https://www.kaggle.com/datasets/usdot/flight-delays Accessed: 18 Feb, 2024.

[54] Rasmus Vestergaard, Daniel E. Lucani, and Qi Zhang. 2020. A Randomly Accessible Lossless Compression Scheme for Time-Series Data. In *IEEE INFOCOM*. https://doi.org/10.1109/infocom41043.2020.9155450

[55] Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani. 2019. Generalized Deduplication: Bounds, Convergence, and Asymptotic Properties. In *IEEE Global Communications Conference (GLOBECOM)*. IEEE. https://doi.org/10.1109/globecom38437.2019.9014012

[56] Rasmus Vestergaard, Qi Zhang, and Daniel E. Lucani. 2019. Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data. In *European Wireless*.

[57] Rasmus Vestergaard, Qi Zhang, Márton Sipos, and Daniel E. Lucani. 2021. Titchy: Online Time-series Compression with Random Access for the Internet of Things. *IEEE Internet of Things Journal* 8, 24 (2021), 17568–17583. https://doi.org/10.1109/jiot.2021.3081868

[58] Sepanta Zeighami, Cyrus Shahabi, and Vatsal Sharan. 2022. NeuroSketch: Fast and Approximate Evaluation of Range Aggregate Queries with Neural Networks. *Proceedings of the ACM on Management of Data* 1, 1 (2022), 1–26. https://doi.org/10.1145/3588954

[59] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. 2014. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 277–288. https://doi.org/10.1145/2588555.2588579

[60] Jiaxing Zhang, Ying Yan, Liang Jeff Chen, Minjie Wang, Thomas Moscibroda, and Zheng Zhang. 2014. Impression Store: Compressive Sensing-based Storage for Big Data Analytics. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. USENIX Association.

[61] Meifan Zhang and Hongzhi Wang. 2021. LAQP: Learning-based approximate query processing. *Information Sciences* 546 (2021), 1113–1134. https://doi.org/10.1016/j.ins.2020.09.070

[62] Meifan Zhang and Hongzhi Wang. 2021. Selectivity estimation with density-model-based multidimensional histogram. *Knowledge and Information Systems* 63, 4 (2021), 971–992. https://doi.org/10.1007/s10115-021-01547-7