FAINDER: A Fast and Accurate Index for Distribution-Aware Dataset Search

Lennart Behme
BIFOLD & TU Berlin
l.behme@tu-berlin.de

Sainyam Galhotra
Cornell University
sg@cs.cornell.edu

Kaustubh Beedkar
IIT Delhi
kbeedkar@cse.iitd.ac.in

Volker Markl
BIFOLD, TU Berlin & DFKI
volker.markl@tu-berlin.de

ABSTRACT

Efficient data discovery is crucial in the era of data-driven decision-making. However, current practices face significant challenges due to the intricacies of identifying datasets with specific distributional characteristics, such as percentiles, when data repositories are decentralized. Traditional keyword-based search methods are insufficient for these complex requirements, often resulting in suboptimal dataset search results. To address these challenges, this paper presents FAINDER, a fast and accurate index for "percentile predicates" on histogram-based data summaries, which streamlines the search process for datasets with specific distributional requirements. FAINDER can be constructed on heterogeneous histogram collections and employs binary search in conjunction with multi-step pruning techniques to efficiently identify search results for percentile predicates. Thereby, it simplifies data provisioning and improves the effectiveness of dataset discovery. Empirical evaluation of our solution on three large-scale data repositories shows that FAINDER is effective for distribution-aware dataset search and provides order-of-magnitude efficiency gains over baselines.

PVLDB Reference Format:
Lennart Behme, Sainyam Galhotra, Kaustubh Beedkar, and Volker Markl. FAINDER: A Fast and Accurate Index for Distribution-Aware Dataset Search. PVLDB, 17(11): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/lbhm/fainder.

1 INTRODUCTION

In today's data-driven world, where organizations collect vast amounts of information from various sources, efficient and effective data discovery has become indispensable. The increasing importance of data discovery is primarily driven by the growing popularity of machine learning techniques, which require substantial volumes of data. Consequently, this trend has led to a surge in data sharing and trading within and across organizations [4, 33].

However, most data discovery systems (and hence data sharing platforms) have limited utility due to two critical design choices that we summarize in Table 1. First, these systems typically assume that all datasets are *completely accessible to the search algorithm* for indexing and processing [12, 13, 23, 25]. This assumption neglects the distributed nature of data repositories. Moving data to a central server is often infeasible due to cost reasons and data owners' reluctance to relinquish control over their proprietary datasets. Instead, a widely adopted data-sharing paradigm involves a distributed collection of data repositories, where each data provider only shares the *metadata* with a search engine. This is commonly observed in data market platforms, such as Datarade [19] and Dawex [20], as well as federated data settings, such as Gaia-X [10] and Agora [54]. Second, existing systems generally depend on users conducting keyword search along with filters or providing example data to find relevant datasets [12, 45]. Even though this search mode is intuitive, it limits discovery in scenarios where users have specific data distribution requirements. For example, users training a machine learning model might seek datasets with a substantial number of samples from each target group to avoid overfitting. More broadly, lack of access to a representative sample for data analysis (also known as selection bias) leads to flawed and unreliable outcomes. These challenges are particularly evident in industrial settings due to repurposing or reusing data [7, 17, 26, 63]. Several data science pipelines have failed because of poor representation of training data [18, 40, 51, 53].

Table 1: Overview of query-driven dataset search approaches, categorized by search mode and data access model.

	Data access model	
Search mode	Full access	Metadata access
Keyword	[12, 13]	[45, 46, 61]
By example‡	[8, 9, 12, 13, 50, 52]	-
Distribution-aware	[3, 14, 41]	FAINDER

‡ This includes finding joinable and unionable tables based on an input table.

Together, the full data access assumption and restricted search mode form a critical roadblock in developing practical data discovery systems. We demonstrate this with the following example.

EXAMPLE. *Consider a data scientist training a cancer prediction model. After developing a prototype, they want to test the robustness of their solution with data from similar trials at other hospitals. To qualify for their work, a dataset must cover different patient ages, so at least 30% of patients should be younger than 40 and at least 30% older than 60. Since such studies contain sensitive information, accessing them requires approval. Therefore, the datasets are not centrally gathered but hosted by the organizations that own each dataset. Consequently, the scientist must search through the publicly available metadata of datasets across several independent data repositories.*

While the status quo for search over decentralized data provides basic functionality, it fails to address the nuanced complexities of searching datasets with distributional requirements. In the example above, our medical scientist only has two options when using keyword-based dataset search: (1) either pose a general keyword query (e.g., "cancer") and manually review a large number

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:XX.XX/XXX.XX

of datasets or (2) add more keywords to the query, hoping that all those keywords are included in the dataset description so that no relevant dataset is filtered out. As a consequence of these functional limitations, users often face too many, off-topic, or no results at all.

In this work, we study the novel problem of distribution-aware dataset search over decentralized data repositories, which complements existing search paradigms. To address the problem, we must tackle three main challenges: (C1) We must develop easily adoptable methods for searching over decentralized datasets; (C2) users must be able to express distributional requirements in search queries; and (C3) search engines must identify datasets that satisfy distributional requirements accurately and efficiently at scale. To address all these challenges, we propose Fainder, an index for distribution-aware data discovery without raw data access, and a new query model.

(C1) Dataset profiles for search over decentralized data. Similar to existing data discovery systems, Fainder assumes that data providers share a profile for each of their datasets with a search engine. To lower the barrier to enriching existing dataset profiles with distribution-aware data synopses, a solution must require as little additional information and effort beyond the status quo as possible. One of the most widespread and simple yet flexible synopses are histograms [16]. They are easy for data owners to generate and seamlessly integrate into dataset profiles. Some dataset search engines, such as Auctus [12] or Kaggle [32], already present histograms as a visual synopsis to the user (but do not use them for distribution-aware search). Fainder allows each data owner to create histograms of their data independently, as it is robust to heterogeneous histograms. Thus, they may individually choose the histogram granularity according to their data privacy sensitivity. If a data owner refuses to provide histograms for a dataset, the search engine can always fall back to only using existing search techniques.

(C2) Percentile predicates for specifying user requirements. We introduce a new type of search predicate that we call *percentile predicate* to offer a simple and intuitive way of specifying distributional requirements. Abstractly, a percentile predicate requires that the dataset values from a given range must or must not represent more than a certain percentage of all values. When composing multiple predicates, this allows users to approximate entire statistical distributions, such as a normal distribution. To integrate with prior work, we propose a simple query model based on Boolean algebra that enables searching for datasets by seamlessly combining existing keyword-based techniques with distributional requirements.

(C3) Indexing for accurate and efficient dataset search. Designing an index for dataset search over decentralized data repositories requires taking multiple stakeholders into account: data owners provide heterogeneous histograms, users expect accurate query results at interactive response times, and search engines aim to scale to extensive dataset collections with minimal resource footprint. We present two variants of Fainder to overcome this challenge. Fainder Approx optimizes the execution time and allows users to search with full precision or recall guarantees. Fainder Exact combines these guarantees in a multi-step solution to prune the search space efficiently. Both Fainder variants address histogram heterogeneity by transforming the unique bins of independently generated histograms into a globally aligned bin distribution. Leveraging the aligned bins, Fainder uses binary search at query time to navigate the search space efficiently. Furthermore, we employ clustering to optimize the trade-off between index accuracy and size.

Outline of Contributions. After introducing basic concepts and our notation in Section 2, we make four major contributions.

- We formally define the problem of distribution-aware dataset search on decentralized data repositories and propose a minimalistic yet effective query model that combines existing search techniques (e.g., keywords) and percentile predicates (Section 3).
- We present Fainder (Section 4), an index for percentile predicates that can be constructed on collections of heterogeneous, independently generated histograms (Section 5).
- We introduce two query modes that trade off runtime and result accuracy (Section 6). Fainder Approx achieves sublinear scaling in the number of datasets while offering different trade-offs for precision and recall. Fainder Exact yields exact query results whilst being significantly faster than the state-of-the-art.
- We conduct an extensive experimental evaluation on real-world open dataset collections (Section 7). Our evaluation shows that Fainder Approx is up to more than two orders of magnitude faster than our baselines. In addition, Fainder Exact also is up to 25× faster than the state-of-the-art.

We close our study with a review of related work in Section 8 and an outlook on future research directions in Section 9. In short, we are the first to investigate distribution-aware dataset search over decentralized data repositories, which is a critical step towards making dataset search practical.

2 PRELIMINARIES

We start by introducing basic definitions about decentralized data repositories and presenting the notation we use throughout the paper (summarized in Table 2).

Dataset. A dataset is a collection of related observations organized and formatted in a particular way [15]. In this paper, we focus on tabular datasets that contain at least one numerical column, as they are both widespread and of particular interest when searching for datasets that meet distributional requirements.

Definition (Tabular Dataset). *A tabular dataset D with a schema $\mathcal{A} \equiv \langle A_1,...,A_l \rangle$ consists of l columns and a collection of tuples T such that $t \in T$ contains l values.*

We refer to the i^{th} column of a dataset D as $D[i]$. Each dataset (and its columns) is often accompanied by specific metadata used for dataset search. For example, Google Dataset Search [45] allows searching through textual dataset descriptions, while Auctus [12] also supports search over column names and their types. Other examples of metadata include row counts or spatiotemporal information. We refer to these joint characteristics as dataset profiles.

Definition (Dataset Profile). *A dataset profile P_D is defined as the properties or constraints the tuples in D satisfy.*

A dataset profile may contain any dataset description, such as column headers, row counts, or licenses [2]. We assume that P_D includes column identifiers, histograms of numerical columns, and the unit of measurement of the histogram elements. Histograms are widely used in data management: Database systems, such as PostgreSQL, use them for query optimization [16], while dataset search engines, such as Auctus [12], use them to visualize dataset

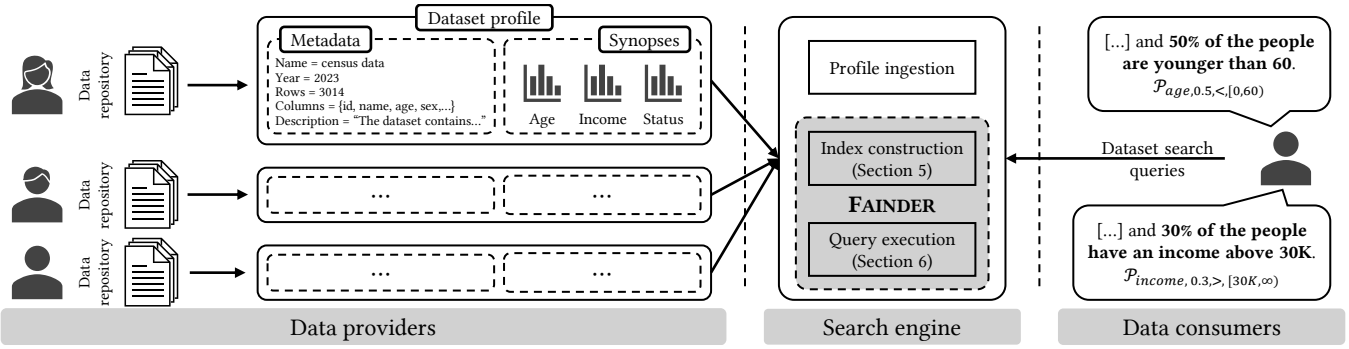


Figure 1: Schematic overview of distribution-aware dataset search and FAINDER's role in it: Data providers offer public dataset profiles used by search engines to answer dataset search queries with distributional requirements effectively and efficiently.

contents. Since data owners can decide for which columns they want to enable distribution-aware search and share histograms, we consider them a low-barrier synopsis for enriching dataset profiles.

Histogram. A histogram is an approximate data synopsis that discretizes a value collection into different bins and stores the frequency of the values that fall into each bin [16]. Without loss of generality, we assume that histograms contain relative frequencies (i.e., densities) and that histogram bins are non-overlapping.

There are numerous ways to compute a histogram on top of the values in a dataset [16]. Our setting is agnostic to the details of histogram creation, as dataset search engines must handle a heterogeneous set of dataset profiles provided by data owners. Consequently, we define a histogram $H_{D[i]}$ over a column $D[i]$ as a list of tuples $\langle b,v\rangle$, where bin $b \equiv [b_l,b_h)$ denotes an interval over the values in $D[i]$ and v denotes the density of values in b. We use `edges` ($H_{D[i]}$) to refer to the list of all bin edges for the histogram $H_{D[i]}$ and `density` ($H_{D[i]}$) for the corresponding list of densities. The subscript $D[i]$ is ignored whenever it is clear from the context.

Data Repository. We consider a setting where different entities (often called data providers) have their own set of datasets. A collection of datasets $\mathcal{D} = \{D_1,...,D_k\}$, gathered by one or multiple data providers, is called a data repository. Each data repository independently computes synopses (histograms for our setting) along with other metadata of its datasets and shares them with a search engine. To support dataset search, the search engine processes these synopses, collates a collection of histograms $\mathcal{H} = \{H_1,...,H_n\}$, and evaluates user queries with it to identify relevant datasets.

3 DISTRIBUTION-AWARE DATASET SEARCH

We formalize the problem of distribution-aware dataset search over decentralized data repositories and discuss its research challenges.

3.1 Problem Definition

Figure 1 provides a semantic overview of the distribution-aware dataset search setting. Based on the data market terminology by Asudeh and Nargesian [3], we consider the passive-provider model, where data owners provide information about their datasets to search engines, and users come to a search engine to find relevant datasets. We assume that users have complex search needs consisting of multiple requirements about the data they seek. Specifically, we focus on distributional requirements. In the following, we introduce our query model and formalize the search problem.

Table 2: Notation overview.

Symbol	Meaning	Symbol	Meaning	Symbol	Meaning
D	Dataset	b	Histogram bin	n	# histograms
$\mathcal{D}$	Dataset collection	c	Cluster	$\mathcal{I}$	FAINDER index
P_D	Dataset profile of D	k	# clusters	Q	Query
H	Histogram	$\mathcal{B}$	Bin budget	$\mathcal{K}$	Keyword pred.
H'	Aligned histogram	$\mathcal{B}_c$	Bin budget for c	$\mathcal{P}$	Percentile pred.
$\mathcal{H}$	Histogram collection	$\mathcal{H}_c$	Histograms in c	S	Result set

Predicate. We model a user's search query as a Boolean predicate that can be checked with respect to any dataset or its corresponding profile. Specifically, we consider *keyword-based predicates*, denoted by $\mathcal{K}_w(D)$, that hold when D's profile contains the specified keyword w. For example, the predicate "*title* = 'census' " returns true for all datasets whose title field contains "census". In contrast, the simpler predicate "cancer" returns true whenever the word "cancer" appears anywhere in the dataset profile. To enable distribution-aware dataset search, we additionally consider percentile predicates.

A *percentile predicate* specifies the condition for a specific distributional requirement. We define the predicate using the notions of *column identifier* C, *fraction* $0 < p \leq 1$, *comparison operator* $\theta \in \{<,\leq,>,\geq\}$, and *range* $r \equiv [r_l,r_h)$, where $r_l,r_h \in \mathbb{R}$ and $r_l < r_h$. For a given C, p, θ, and r, the predicate $\mathcal{P}_{C,p,\theta,r}(D)$ defined on a dataset D holds whenever the dataset contains a column matched by C and the comparison $p\,\theta\,f$ is true, where f is the fraction of values in the column that lie within the range $[r_l,r_h)$. Other range definitions are possible but omitted for the sake of brevity. More formally:

$$\mathcal{P}_{C,p,\theta,r}(D) := \begin{cases} \text{true} & \text{if } \exists D[i]: \\ & (D[i] \in C) \wedge \left(p\,\theta\,\frac{|\{x : x \in D[i] \wedge x \in [r_l,r_h)\}|}{|D[i]|}\right) \\ \text{false} & \text{otherwise} \end{cases}$$

The column identifier C can be a simple string or a more complex operation, such as a column match based on semantic similarity [61]. In simple words, the predicate $\mathcal{P}_{\text{age},0.5,\leq,[0,60)}(D)$ holds if *"dataset D has a column 'age' and at least 50% of the people are younger than 60"*. However, a key challenge of evaluating percentile predicates over decentralized data repositories is that the search engine only has access to dataset profiles instead of raw data. Therefore, we must design solutions capable of evaluating $\mathcal{P}(P_D)$ rather than $\mathcal{P}(D)$.

Query. A query Q is a Boolean combination of keyword and percentile predicates. Keyword-based dataset search has been extensively studied in the literature [15, 45, 57, 61]. Here, we focus on supporting distribution-aware search with $Q(D) \equiv \mathcal{P}(D)$.

We now define the distribution-aware dataset search problem.

Problem (Distribution-Aware Dataset Search). *Given a collection of data sources $\mathcal{S}$, such that each source s has a data repository $\mathcal{D}_s = \{D_1,...,D_d\}$ and shares corresponding dataset profiles $\mathcal{M}_s = \{P_{D_1},...,P_{D_d}\}$, and a dataset search query Q, find the set of datasets $S = \{D : D \in \mathcal{D}$ and $Q(P_D)$ holds$\}$.*

3.2 Research Challenges

Of the three challenges presented in Section 1, we address C1 by using histograms and C2 by proposing percentile predicates. We now discuss the difficulties of designing a fast and accurate solution for evaluating percentile predicates on histograms (C3). Across all stakeholders, the desiderata for a distribution-aware dataset search engine are: (1) interactive query execution time with sublinear scaling in the number of datasets, (2) accurate results, and (3) a minimal resource (i.e., memory) footprint of any data structures other than dataset profiles. These three dimensions form a triangle with a performance trade-off since we cannot maximize all dimensions simultaneously: producing accurate results in minimal time requires precomputation, which implies higher memory usage; accurate results without additional memory usage require an iterative execution with linear runtime; and returning results in sublinear time with minimal memory usage effectively means guessing with arbitrary accuracy. Therefore, at least one dimension will always perform worse than if we solely optimized a solution for it. Beyond this trade-off, a central challenge of our problem is the handling of heterogeneous histograms provided by the data owners.

Predicate Evaluation With Histograms. In principle, evaluating a percentile predicate using a histogram H is straightforward, as its bin edges and densities include the required information. Consider the exemplary histograms from Figure 2 and our predicate $\mathcal{P}_{\text{age},0.5,\leq,[0,60)}$, looking for datasets where at least 50% of the people are younger than 60. To evaluate it, we sum up the density f of all bins that fall into $[r_l,r_h)$. For H_a, we have $f = 0.6$; thus, the predicate holds. Likewise, it is straightforward to see that $\mathcal{P}$ does not hold for H_b. Yet, the triviality of this example does not extend to decentralized data repositories, as each data provider independently creates histograms of their data at will so that a bin may only partially overlap with $[r_l,r_h)$, such as in the case of histogram H_c.

Evaluating a predicate on heterogeneous histograms often requires estimating a bin's contribution to f. There are several intra-bin value distribution estimation strategies [16]. Common strategies are to underestimate (ignore bin b), overestimate (add the total density of b), or assume a continuous value distribution and add a share of b's density proportional to the overlap with range r. Unless we overestimate f for $\theta =$ "$<$" and underestimate it for $\theta =$ "$>$", we risk filtering out dataset columns that match the predicate. Since a histogram is a lossy representation of D, this approach may produce false positives (a limitation of synopsis-based dataset search). However, the approach guarantees full recall, a crucial requirement to ensure effectiveness. If full recall is not required, the estimation could be changed to achieve full precision or to maximize F_1 score.

Predicate Evaluation Over Dataset Collections. A simple yet effective way to evaluate a percentile predicate at the dataset collection level is a technique we call `profile-scan`. It iterates through each histogram H, determines the bins that fall into the range r, and

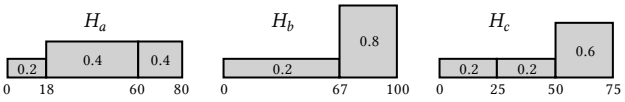


Figure 2: Histograms H_a, H_b, and H_c.

adds H to the result set S if the predicate is fulfilled. This makes it an accurate and memory-efficient solution, as it uses no additional data structures. However, a major limitation is that it scales linearly in the number of histograms. Even though `profile-scan` can filter the histogram collection based on column identifier C, a significant number of histograms can remain after filtering. For example, Kaggle [32] has more than 15000 datasets that match the keyword "age" and more than 23000 datasets matching the query "date" as of February 2024. This problem becomes exacerbated if users pose generic column identifier predicates instead of keywords, such as *"match any column related to finance"*. Thus, `profile-scan` leaves a need for a fast and accurate as well as a fast and resource-efficient solution, which we address in this paper.

The critical problem for the sublinear scalability of percentile predicate evaluation is the heterogeneity of histograms from decentralized data repositories. As the histograms have different numbers and distributions of bins, we must individually identify the bins within a predicate's range per histogram. Allowing arbitrary value ranges in a predicate further complicates the problem since an infinitely large number of possible ranges must be considered for an index. If we instead require that either $r_l = -\infty$ or $r_h = \infty$, we can achieve orders of magnitude speedups for percentile predicates with intelligent precomputation. Such "one-sided" intervals for specifying a range enable us to execute queries similar to the ones in our motivating example. Assuming that $r_l = -\infty$ or $r_h = \infty$, the range r of a predicate effectively cuts the number line into two parts. Consequently, we can rewrite any predicate with $r_l = -\infty$ into a predicate with $r_h = \infty$ (and vice versa) by setting $r_l = r_h$, replacing p with $1-p$ and flipping the comparison operator θ. Without loss of generality, we therefore assume $r_l = -\infty$ and simplify the notation of a percentile predicate to $\mathcal{P}_{C,p,\theta,r_h}(P_D)$ for the rest of this paper.

4 FAINDER

We propose Fainder, an index for percentile predicates that attains robust scalability and accurate (optionally exact) results to address the identified research challenges. This section gives an overview of Fainder's architecture. Subsequently, we discuss the technical details of index construction and querying in Sections 5 and 6.

Key Insights. The central intuition behind Fainder is that whenever all histograms have the same set of bins, we can precompute a subset of cumulative densities to answer percentile predicates efficiently with sublinear time complexity. Conceptually, we achieve a logarithmic scaling by sorting the histograms' cumulative densities at each bin boundary and using a two-stage binary search over the bins and densities to identify the result set. In practice, however, histogram heterogeneity prevents us from directly using binary search on a collection. To overcome this limitation, Fainder homogenizes histograms by mapping them to a collection-wide consistent bin distribution. Changing the bin edges of a histogram and reassigning a bin's density to one or multiple new bins is inherently approximate. Thus, we must ensure that Fainder has high accuracy comparable to `profile-scan` during index construction. Naive approaches to

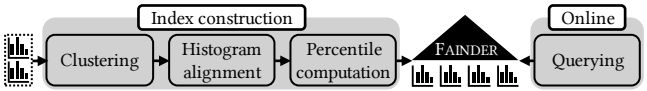


Figure 3: Overview of FAINDER.

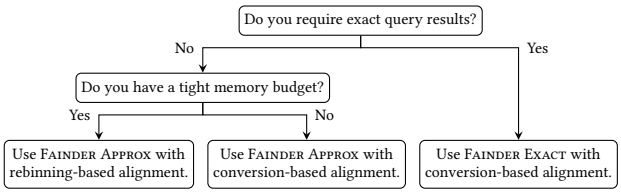


Figure 4: Flowchart for choosing a FAINDER variant.

creating such a global distribution are constructing the union of all original bin edges or creating ϵ-sized bins between the minimum and maximum value of a collection. However, these approaches lead to an excessive memory footprint or subpar result accuracy, as we show with a concrete example in Section 5.1. FAINDER addresses these challenges by creating a clustered global bin distribution instead. Empirical evaluation on one million datasets (GitTables [29]) shows that our clustering approach reduces the memory footprint from more than one TB to 3.2 GB without a substantial increase in runtime while maintaining high accuracy.

System Overview. Figure 3 summarizes the lifecycle of FAINDER. The starting point is a histogram collection $\mathcal{H}$ and a mapping $\texttt{column}(H)$ that projects each histogram to its associated column identifier. At first, FAINDER clusters $\mathcal{H}$ based on the bin edges of each histogram. As part of the clustering phase, we must decide what share of the global bin budget (i.e., memory budget) we assign to each cluster and how the bin edges within a cluster should be distributed. Next, FAINDER aligns the histograms $\mathcal{H}_c$ in a cluster c by distributing the original bin density $\texttt{density}(H)$ to $\texttt{density}(H')$ based on the cluster bins $\texttt{edges}(\mathcal{H}_c)$. For this, we introduce two alternative histogram alignment techniques, rebinning and conversion, that offer different trade-offs for precision, recall, and index size. Having aligned all histograms in a cluster, FAINDER indexes them by precomputing and sorting the cumulative density per histogram and bin. Finally, FAINDER evaluates percentile predicates using binary search at query time. Like any search index, FAINDER amortizes its additional memory consumption and one-time construction cost through repeated execution time savings each time it evaluates a predicate. We discuss the conceptual index maintenance costs in Section 5.3 and experimentally evaluate them in Section 7.4.

Index Variants. The two histogram alignment techniques enable us to design three unique FAINDER variants that cater to different requirements. Figure 4 summarizes them with guidelines for choosing an appropriate version. If exact results are not required or execution time is paramount, rebinning and conversion offer two alternative FAINDER APPROX variants. While rebinning minimizes the index size and construction time, conversion allows users to select a full precision or recall guarantee at query time. Thus, FAINDER EXACT utilizes a conversion-based index to efficiently prune the search space before conducting `profile-scan` on a small subset of histograms. Note that a conversion-based index can be used for both approximate and exact results based on the runtime preference per query. Therefore, the primary decision during index construction is to choose either rebinning- or conversion-based histogram alignment.

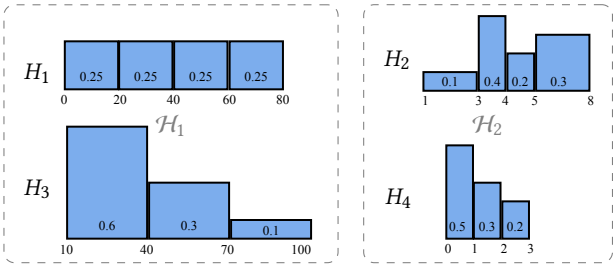


Figure 5: Histograms H_1–H_4 and clusters $\mathcal{H}_1$,$\mathcal{H}_2$.

5 INDEX CONSTRUCTION

We discuss the construction of FAINDER, which consists of the steps clustering, histogram alignment, and percentile computation.

5.1 Clustering

The clustering phase computes a feature vector for each histogram, clusters them, and defines the aligned bins for each cluster. The motivation for this phase are the bin width and value range differences between histograms in a dataset collection. To illustrate this, consider the histogram collection from Figure 5, which we use as a running example throughout this section, and assume that there is another histogram H_5 with $\texttt{edges}(H_5) = [-100,-85,-70]$ (not depicted for brevity). The global value range of this collection is [-100, 100]. However, no histogram covers the interval (-70, 0). Therefore, allocating bins in the global bin distribution for this region would be a waste of space, as there are no density changes. Furthermore, H_1 and H_4 both cover the interval [0, 3] but with vastly different bin widths. Thus, converting $\texttt{edges}(H_1)$ and $\texttt{edges}(H_4)$ to a joint bin distribution would be too detailed for H_1 or too coarse for H_4. Considering that FAINDER should have a minimal memory footprint, aligned bins must only cover relevant parts of the global value range with a locally appropriate bin width to achieve accurate query responses efficiently. Since we do not know user queries in advance, an optimal solution is query-agnostic and minimizes the information loss between original and global bins, given a user-defined bin budget $\mathcal{B}$. Formally, we consider the overlap of each original bin $b_i \equiv [b_{il}, b_{ih})$ with the closest global bin b^g_j as a proxy for mitigating information loss and thus aim to maximize it:

$$\underset{b^g}{\texttt{argmax}}\ O = \sum_i \max_j \left(\frac{\max\left(\texttt{min}(b_{ih}, b^g_{jh}) - \texttt{max}(b_{il}, b^g_{jl}), 0\right)}{\max\left(\texttt{width}(b_i), \texttt{width}(b^g_j)\right)} \right) \quad (1)$$

The cumulative overlap O reflects the overlap between an original bin and the most closely fitting global bin (red), normalized by the bin width (blue), and summed across original bins (brown). If we consider each possible global bin as a unit-weight item whose value is its marginal improvement of O, finding an optimal solution with a limited number of bins is a variation of the knapsack problem (i.e., NP-hard). Therefore, we use a three-step procedure based on clustering to break down the global bin definition problem into smaller parts and efficiently find an approximate solution.

Clustering Features. Clustering aims to partition the histogram collection $\mathcal{H}$ into sets of histograms that cover similar value ranges with similar bin widths. Therefore, we compute a feature vector

$$v_H = \left[\texttt{min}(\texttt{edges}(H)) \quad \texttt{max}(\texttt{edges}(H)) \quad \texttt{avgWidth}(\texttt{edges}(H))\right]$$

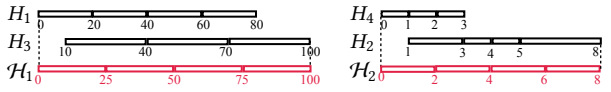


Figure 6: Cluster bin assignment for $\mathcal{H}_1$ and $\mathcal{H}_2$ ($\mathcal{B}=8$).

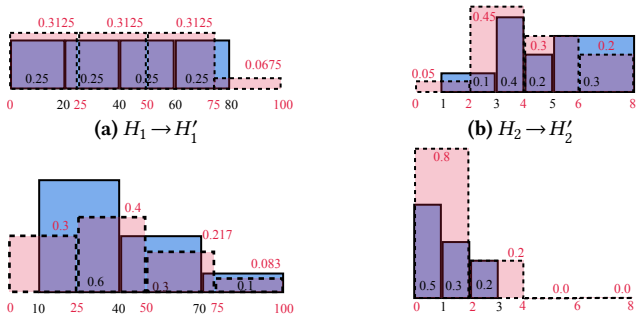


Figure 7: Rebinning-based histogram alignment. The aligned histogram bins and densities are shown in red.

for each histogram, where `avgWidth` represents the average bin width of a histogram. Due to the heterogeneity of histograms, directly clustering the feature vectors would yield suboptimal results. Therefore, we preprocess them with a non-linear quantile transform [49], which maps all features to a uniform distribution on the interval [0,1] to reduce the impact of outliers. The intuition for this transformation is to make the different value scales of our features more directly comparable while being robust to outliers.

Clustering Algorithm. Fainder requires a scalable clustering algorithm to partition all histograms into reasonable, ideally balanced clusters. Conceptually, the clustering algorithm choice is a hyperparameter of Fainder and orthogonal to the other stages. Concretely, we use k-Means [37] for clustering and discuss our investigation of alternative clustering algorithms in Section 7.3.

Cluster Bin Assignment. After clustering, we must distribute the global bin budget $\mathcal{B}$ across clusters. Figure 6 demonstrates this final step of the clustering phase with our running example. The simplest method is to assign a budget proportional to the cluster size:

$$\mathcal{B}_c = \mathsf{max}\left(1, \left\lfloor \mathcal{B} \frac{|\mathcal{H}_c|}{\sum_{i=1}^{k} |\mathcal{H}_i|} \right\rfloor\right) \tag{2}$$

However, in the worst case, this results in small clusters only consisting of a single bin, whereas large clusters might receive a much more fine-grained bin width than needed. Therefore, we use additive smoothing [38] to anneal the proportional bin budget towards a uniform assignment based on the parameter α. Given a cluster (value range) and bin budget, we create equi-width bins $\mathtt{edges}(\mathcal{H}_c)$ for each cluster. Conceptually, the cluster bin assignment strategy is a hyperparameter that a search engine operator can adapt. For example, we could instead allocate the same bin budget to each cluster or choose an individual bin definition algorithm for each cluster, such as equi-height bins (which are more costly to compute).

5.2 Histogram Alignment

Given a cluster of histograms $\mathcal{H}_c$ and its cluster bins $\mathtt{edges}(\mathcal{H}_c)$, the task of histogram alignment is to transform each $H \in \mathcal{H}_c$ to H', such that $\mathtt{density}(H')$ represents the original value distribution of H on $\mathtt{edges}(\mathcal{H}_c)$. This requires distributing the density of each original bin to one or multiple new bins. Consequently, histogram alignment requires approximating the data distribution within a bin and strongly influences the downstream index accuracy.

We present two alternative approaches for histogram alignment that offer different trade-offs between index accuracy and resource requirements. Rebinning aims to minimize the index size but does not provide a formal accuracy guarantee. It allows the system operator to make an assumption about the intra-bin value distribution and will balance precision and recall if the assumption holds. While rebinning is an intuitive and efficient way to align histograms, there are scenarios where guaranteeing total precision or recall is critical. For example, precision is essential if users only want to review a few guaranteed true results. Contrarily, total recall is crucial to use Fainder as a pruning tool for an exact solution. Thus, conversion ensures total recall or precision at query time but requires additional storage compared to rebinning.

Rebinning. The principal idea of rebinning is to compute the overlap interval $o_{ij} = [\mathsf{max}(b_{il}, b_{jl}), \mathsf{min}(b_{ih}, b_{jh}))$ between all pairs of bins b_i from the original histogram and b_j from the respective cluster bins. Given o_{ij}, we compute the fraction of the original bin's density $\mathtt{density}(H)[i]$ that lies within o_{ij} and add it to $\mathtt{density}(H')[j]$. Since a histogram does not contain information about the distribution of values within a bin, rebinning must approximate this fraction if $o_{ij} \neq b_i$. The simplest intra-bin density estimation assumes a uniform value distribution within a bin (i.e., we add $\frac{\mathtt{width}(o_{ij})}{\mathtt{width}(b_i)}$% of b_i's density to b_j) [16]. However, a search engine can make other assumptions about the data distribution. For instance, cubic spline interpolation is a more costly alternative but can better estimate non-uniform value distributions.

Figure 7 continues our running example and shows how to rebin H_1–H_4 based on the cluster bins. For example, the fourth bin of H_2 has the range [5,8]. Thus, its overlap with $\mathcal{H}_2[2]$ is $o_{32} = [5,6)$. Under a continuous value assumption, this means that one-third of $H_2[3]$'s density (0.1) is assigned to $\mathtt{density}(H'_2)[2]$, which, together with the contributions from $H_2[2]$, has a total density of 0.3.

Conversion. The central intuition of conversion-based histogram alignment is to compute a lower and upper bound for the density at each bin. To compute these bounds, we create a *conversion matrix* CM for every array pair ($\mathtt{edges}(H)$, $\mathtt{edges}(\mathcal{H}_c)$) of original bins and cluster bins. CM is a Boolean matrix that is true for each bin pair that partially or fully overlaps. Based on CM, we know which original bin *might* or *must* contribute to the density of any bin in H'. Since the original and cluster bins form an n:m relationship, we cannot assume that the upper bound of $H'[i]$ is the lower bound of $H'[i+1]$. This requires us to store two numbers per bin, resulting in a 2× index size compared to rebinning. At the same time, conversion enables predicate evaluation with full precision or recall, as we do not perform any intra-bin estimation but fully add $\mathtt{density}(H)[i]$ to $\mathtt{density}(H')[j]$ if it contributes to the lower or upper bound of the bin's density. Depending on the comparison operator in a predicate, Fainder uses the respective density bound to prevent false positives or negatives. Given a query, such as "at least 50% of the values are lower than 60" with a total recall requirement, the upper density bound lets us filter out those histograms that do not have a cumulative density higher than 50% in the bin where 60 lies.

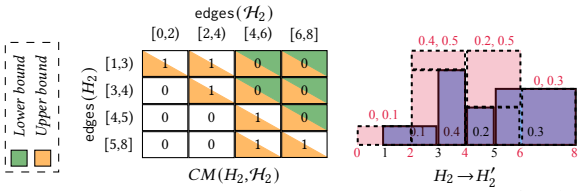


Figure 8: Conversion-based histogram alignment: (left) conversion matrix for computing percentile bounds, (right) aligned histogram with lower and upper bounds in red.

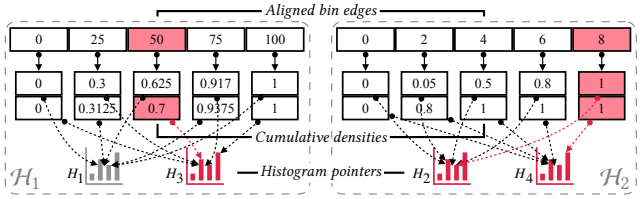


Figure 9: Rebinning-based FAINDER index for H_1–H_4. Red color outlines evaluating the predicate *"at least 65% of the values are less than 50"*. Dashed arrows are histogram pointers.

Figure 8 shows the conversion matrix for histogram H_2 and cluster $\mathcal{H}_2$ on the left. On the right, we visualize the bin-wise conversion of H_2. In practice, we do not convert each bin separately but directly compute the lower and upper bounds for the cumulative density of a bin. By knowing which cluster bins an original bin partially contributes to, we also know that this bin must fully contribute to the cumulative density of all following cluster bins, highlighted by the green lower bound area. The orange upper bound area consists of the lower bound area and the bins whose contribution is unclear. We compute the cumulative bounds by summing up the density of all original bins that contribute to a cluster bin's lower or upper bound. The cumulative density of $H_2'[2]$, for example, is bounded by $[0.5,1]$ since the first two bins of H_2 (green area) fully contribute to the bin's density. In contrast, the orange area (bins 1-4) comprises the original bins that might contribute to the cumulative density of $H_2'[2]$.

5.3 Percentile Computation

The percentile computation phase consists of initialization, density summation, and sorting. We demonstrate its result in the upper part of Figure 9. For each cluster c, FAINDER creates a $(n_c \times \mathcal{B}_c)$ array to store percentiles and a pointer array of the same size to keep a reference to the respective histograms. For conversion-based indices, this array pair is created twice for the lower and upper percentile bounds. Next, FAINDER computes the cumulative density per cluster bin and histogram. Since all histograms in a cluster have the same bins, FAINDER uses vectorization to compute the cumulative sums efficiently and stores them in the percentile array. Lastly, we sort the percentile array and corresponding pointers column-wise. This way, FAINDER knows that all histogram pointers after a given one in the current bin have an equal or higher cumulative density.

Index Size and Maintenance Costs. FAINDER's query execution time improvements must justify its overhead, specifically the memory consumption and maintenance cost. Without clustering, FAINDER's asymptotic space complexity is $O(n\mathcal{B})$, as we must store a percentile-pointer pair for each histogram and global bin. However, with clustering, we do not have to process the full Cartesian product of histograms and bins across clusters, making construction faster and the index smaller. The more clusters an index has, and the more evenly balanced they are, the smaller the size of FAINDER. In the case of a perfectly even distribution, the size shrinks to $O(\frac{n\mathcal{B}}{k})$.

Algorithm 1: Percentile predicate evaluation with FAINDER.

Input: index $\mathcal{I}$, cluster bins $\{\text{edges}(\mathcal{H}_c)\}_{c=1}^{k}$, predicate $\mathcal{P}$
Output: Solution set S

1 $C,p,\theta,r_h \leftarrow \mathcal{P}$, $S \leftarrow \{\}$, $l \leftarrow 0$
2 **if** $\theta \in \{<,\leq\}$ **then** $l \leftarrow 1$ // Upper bound for "at least" predicates
3 **for** $c \leftarrow 1$ **to** k **do**
4 **if** $\min(\text{edges}(\mathcal{H}_c)) \leq r_h \leq \max(\text{edges}(\mathcal{H}_c))$ **then**
5 $i \leftarrow \text{binarySearch}(\text{edges}(\mathcal{H}_c), r_h)$ // Bin index i
6 $j \leftarrow \text{binarySearch}(\mathcal{I}_{cl}^{P}[:,i], p)$ // Histogram index j
7 **if** $\theta \in \{<,\leq\}$ **then** // "at least" predicate
8 $S \leftarrow S \cup \mathcal{I}_{cl}^{H}[j:,i]$ // Include all rows after j
9 **else** // "at most" predicate
10 $S \leftarrow S \cup \mathcal{I}_{cl}^{H}[:j,i]$ // Include all rows before j
11 **else** // r_h not in cluster range
12 **if** $(r_h \leq \min(\text{edges}(\mathcal{H}_c)) \wedge \theta \in \{>,\geq\}) \vee$
 $(r_h \geq \max(\text{edges}(\mathcal{H}_c)) \wedge \theta \in \{<,\leq\})$ **then**
13 $S \leftarrow S \cup \mathcal{I}_{cl}^{H}$ // Add all pointers from cluster
14 **foreach** $s \in S$ **do**
15 **if** $\text{column}(s) \notin C$ **then** $S \leftarrow S - s$
16 **return** S

The maintenance costs of FAINDER are divided into one-time construction costs and repeated histogram insertion and deletion costs. Since we have to initially cluster and individually align each histogram, the construction costs scale linearly with n. However, these costs are mitigated by the fact that an index is seldom created but often queried. More importantly, we can insert into and delete from FAINDER at minimal cost since histogram alignment is an incremental process as long as we do not change the cluster bins and assign new histograms to existing clusters.

6 INDEX QUERYING

We discuss using FAINDER's design for fast and accurate percentile predicate evaluation. Our index offers two query modes, FAINDER APPROX and FAINDER EXACT, which we delineate in this section.

6.1 FAINDER APPROX

The approximate version of our index is designed to minimize runtime while offering two different approaches to trade off resource efficiency and accuracy guarantees. Algorithm 1 shows the query procedure for a conversion-based index with full recall guarantees. Formally, FAINDER is a data structure $\mathcal{I} \equiv (\mathcal{I}^P, \mathcal{I}^H)$ that consists of percentile and histogram pointer arrays. We use $\mathcal{I}_{cl}$ to denote the cluster $c \in \{1,\ldots,k\}$ and the lower or upper percentile limit $l \in \{0,1\}$. FAINDER first decides which bound to use based on the predicate's comparison operator θ. For a rebinning-based index, l is always 0. Then, it iterates through each index cluster c. If the query's range value r_h does not fall into the range of a cluster, FAINDER adds either all or no histograms from the cluster to S, depending on θ and whether r_h lies above or below the cluster range. Otherwise, FAINDER first performs binary search to identify which cluster bin r_h falls into (line 5). Then, it conducts another binary search within that bin to identify the first (for "at least" predicates) or last (for "at most") histogram that matches p (line 6). Finally, it adds the histogram pointer from the identified cell and all following ("at least",

line 8) or preceding ("at most", line 10) pointers from the column to S. The query procedures for rebinning- or conversion-based indices with total precision differ from Algorithm 1 in a few conditional statements that we omit for improved readability.

Figure 9 concludes our running example by visualizing the predicate *"at least 65% of the values are less than 50"* on our example index (in red). In cluster $\mathcal{H}_1$, H_1 has a cumulative density of only 0.625 at bin edge 50, wherefore it is excluded from the result. For $\mathcal{H}_2$, we can directly append all histograms to the result without performing a binary search since the cluster range implies that 100% of all values must be smaller than or equal to eight.

Asymptotically, the time complexity of Fainder Approx without clustering is $O(\log(n)\log(\mathcal{B})+|S|)$, where $|S|$ denotes the result size. With k clusters of asymptotically similar size, the complexity changes to $O(\log(\frac{n}{k})\log(\frac{\mathcal{B}}{k})k+|S|)$. In Section 7.4, we analyze the practical impact of k and $\mathcal{B}$ on runtime, accuracy, and index size.

6.2 Fainder Exact

While Fainder Approx is a fast and resource-efficient solution, it does not address the requirement of a fast and accurate solution from the performance triangle we discussed in Section 3.2. To fill this gap, we present Fainder Exact, a multi-step extension of our index. Albeit Fainder is an approximate index in its core, we use conversion's recall and precision guarantees to construct an exact solution that is up to 25× faster than `profile-scan` in practice.

Fainder Exact uses Fainder Approx as an effective pruning technique in a three-step solution by leveraging the accuracy guarantees of conversion. First, we use the recall variant of Fainder Approx to filter out most false results while preventing false negatives. Next, we apply Fainder with total precision to identify histograms that definitely are part of the result set. At last, we perform `profile-scan` on the set of results that are part of the full recall but not the full precision result to filter out the remaining false positives. In our evaluation, Fainder Exact had to evaluate an order of magnitude fewer histograms than a full `profile-scan` in the third stage without impairing accuracy. Running a query in total recall as well as total precision mode and comparing the results also benefits the user's search experience without a subsequent `profile-scan`. By computing the set difference between the two results, a search engine can categorize results as "guaranteed" or "potential", giving users more information for choosing which datasets to review manually.

7 EVALUATION

We experimentally evaluate the efficacy of Fainder in three steps. After describing our setup, we first investigate the efficiency of Fainder. Next, we analyze our approach's effectiveness. Lastly, we conduct a set of micro-benchmarks to demonstrate how Fainder's core parameters jointly influence all dimensions of its performance.

Most importantly, our evaluation shows that Fainder Exact dominates our baselines in a skyline analysis while Fainder Approx is up to more than two orders of magnitude faster than them.

7.1 Experimental Setup

We implemented our prototype for Fainder in Python, leveraging performance optimizations from packages like NumPy and scikit-learn [49]. Our experiments used an Ubuntu 22.04 server with two Intel Xeon 6330 CPUs (112 vCores @ 2.0GHz) and 512 GB RAM.

Table 3: Overview of benchmark dataset collections. Size represents the total size of all files in the collection.

Name	ID	# Datasets	Size (GB)	# Histograms
SportsTables [35]	ST	1 183	0.3	19 862
Open Data [23]	OD	5 966	29	68 313
GitTables [29]	GT	1 018 649	39	5 017 619

Dataset Collections. Table 3 summarizes the three real-world dataset collections we used. SportsTables [35] is interesting to analyze because it has been explicitly curated to contain many numeric columns with realistic value distributions. Open Data [23] is an excerpt of datasets from Open Data Portal Watch [44], which aggregates statistics from 280 open data portals, such as NYC Open Data. Lastly, we used GitTables [29] to evaluate the scalability of Fainder. None of the collections currently include dataset profiles with histograms, wherefore we downloaded the raw data and generated histograms ourselves. We randomized the number of bins per histogram to simulate heterogeneous dataset profiles from different data repositories. The histogram value range and average bin width of Open Data and GitTables span more than 15 orders of magnitude.

Baselines. `profile-scan` (Section 3.2) is a natural baseline for our problem. We consider the results of `profile-scan` as the ground truth for a query since there is no way to compute a more accurate answer to a percentile predicate based on histograms. In addition, `binsort` is an optimized baseline that precomputes lower and upper percentile estimates for each bin edge and sorts the percentiles by their bin edge. At query time, `binsort` can use binary search on the bin edge domain but has to perform a linear scan over the results to evaluate the percentile requirement since there is no total sort order over both dimensions. Thus, it represents a middle point between `profile-scan` and Fainder, which is able to leverage binary search on both dimensions. To evaluate the space-efficiency of Fainder, we furthermore introduce `normal-dist`, which approximates each numerical column with a normal distribution and thus only has to store two values per column instead of $\mathcal{B}_c$ values per histogram. Note that while `normal-dist` is space-efficient, it does not achieve a sublinear execution time, as the two parameters of a normal distribution do not have a total ordering in one dimension.

Benchmark Queries. To the best of our knowledge, there is no benchmark for distribution-aware dataset search queries, requiring us to define the set of evaluation queries ourselves. Since there is no widely agreed-upon metric to estimate the "difficulty" of a dataset search query in terms of answering it quickly and accurately, we randomly generated a diverse set of 10 000 queries with percentile predicates and categorized them based on three different metrics: query selectivity (QS) describes the percentage of histograms a query matches (e.g., 0.9 means that 90% of the histograms match a query); the share of cluster matches (SC) measures the number of clusters in an index with which a percentile predicate's value range overlaps; finally, the bin edge matches (BE) count how many original histogram bin edges fall on the end of a predicate's value range. In a preliminary experiment, we analyzed the value distribution of each metric across our query set (see Figure 10 for Open Data). The query selectivity splits the

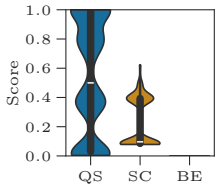
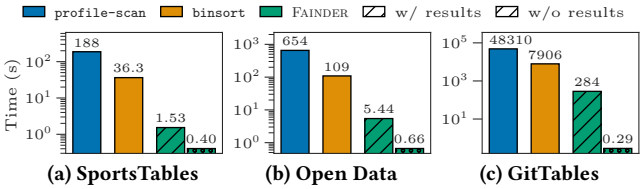
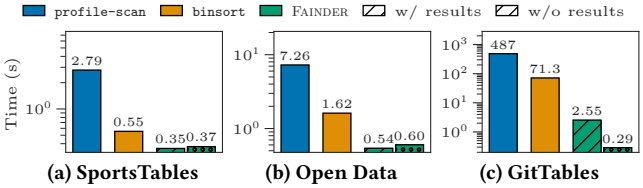


Figure 10: Open Data query metrics.



(a) SportsTables (b) Open Data (c) GitTables

Figure 11: Runtime comparison of `profile-scan`, `binsort`, and Fainder Approx over 999 queries.



(a) SportsTables (b) Open Data (c) GitTables

Figure 12: Runtime comparison of `profile-scan`, `binsort`, and Fainder Approx over 999 queries with low selectivity.

set into subsets with a high (>90%), medium (10-90%), or low (<10%) selectivity, whereas the other two metrics did not prove to be robust query categorizations. Most predicate ranges overlap with 10-40% of the clusters, while the number of exact matches in the original bin edges is zero for next to all queries. Therefore, we focused on the selectivity categorization and randomly sampled 333 queries from each selectivity group to collate a diverse set of 999 benchmark queries. In addition, we sampled another 100 queries per category to obtain a set of 300 queries that we used for a comprehensive grid search (see Section 7.3) to identify the best parameters of our index.

7.2 Solution Efficiency

We start our evaluation of Fainder with an efficiency analysis and present five experiments: a runtime comparison, a scalability analysis, runtime breakdowns of Fainder Approx and Exact, and an index construction time analysis. Since `normal-dist` has no runtime advantage over `profile-scan`, we omit it in this section.

Runtime Comparison. Figure 11 shows an execution time comparison for our 999 test queries. The figure highlights that Fainder Approx is more than two orders of magnitude faster than `profile-scan` and $20-28\times$ faster than `binsort` on all three dataset collections. Furthermore, we observe that Fainder Approx achieves interactive execution times for each collection if we scale down the runtime to an individual predicate evaluation.

As we only benchmark the runtime of individual percentile predicates instead of large composite queries in this experiment, the result set S can become quite large. Therefore, we also ran a modified version of Fainder that performs all steps from Algorithm 1 until line 13 but returns a dummy result of size 1 to filter out the linear time impact of processing the result set. This "without results" execution time demonstrates the potential runtime impact of S. While the difference to the regular execution of Fainder is $4-5\times$ for SportsTables and Open Data, Fainder is another two orders of magnitude faster on GitTables when not returning results. This shows how significant Fainder's runtime advantage over the baselines is if the execution is not dominated by S – either because users run composite queries with multiple different predicates or because a percentile predicate itself has very low selectivity.

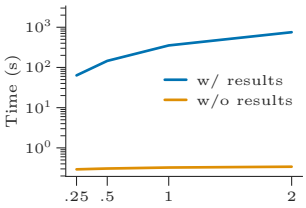


Figure 13: Runtime on GitTables across scaling factors.

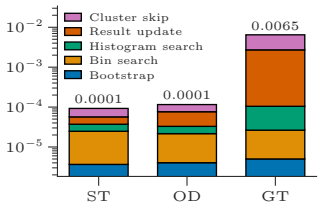


Figure 14: Predicate evaluation runtime breakdown.

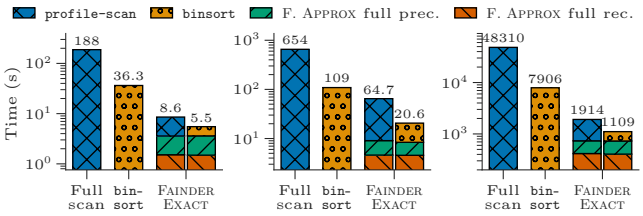
To specifically test the runtime of Fainder when a percentile predicate has very low selectivity, we simulated a restrictive column identifier that only matches 1% of the histograms and then ran our benchmark queries on the resulting histogram subset. The average selectivity of our queries when combining the column identifier and the distributional requirement lies at 0.5%. Figure 12 highlights that Fainder maintains its performance lead over the baselines for low-selectivity queries, while the relative advantage shrinks for the small dataset collections SportsTables and Open Data. This is because Fainder's runtime grows or shrinks logarithmically with the collection size while the baselines benefit linearly from a smaller collection. We also see that the result set S no longer impacts the runtime for the small collections. Regarding GitTables, about 10 000 histograms remain after prefiltering so that Fainder still outperforms all baselines by orders of magnitude, consistent with Figure 11. Overall, the experiment shows that Fainder is a superior choice in terms of runtime, especially for large-scale dataset search.

Scalability. We investigate the scalability of Fainder by creating four versions of our largest collection GitTables, including each histogram 0.25, 0.5, 1, and 2 times on average to achieve a respective scaling factor. Figure 13 summarizes the runtime of Fainder for all scaling factors. To account for the impact of the solution size S, we again conduct the experiment with and without processing the search results. We see that the execution time of the runs that return results increases linearly with the scaling factor since $|S|$ also increases linearly with the scaling factor. Contrarily, the runtime of Fainder is almost constant without returning results, highlighting its logarithmic scaling in the number of histograms and bins.

Fainder Approx. After reviewing the scalability of Fainder, we zoom into the individual phases of predicate evaluation with Fainder Approx. Due to the challenges of tracing a system-under-test without altering its performance [31], we only measure the conceptually necessary steps of Fainder in this experiment. Note that tracing those steps already increases a query's execution time by one order of magnitude due to logging overhead.

Figure 14 dissects the core operations of our index for the exemplary predicate $\mathcal{P}_{*,0.1,<,50}$. The experiment shows that the two most critical operations, bin and histogram search, scale sublinearly with the dataset collection size. While bin search time is almost the same across collections, histogram search only grows by 7× for GitTables, although the collection is 252× and 73× larger than the other two collections. The result set update, on the other hand, scales linearly with $|S|$ and thus takes noticeably more time for GitTables. Cluster skip partially also scales with $|S|$ as all histograms from a cluster might have to be added to S depending on line 12 in Algorithm 1.

Fainder Exact. Building on the analysis of Fainder Approx, we investigate the efficiency of our exact solution. Figure 15 shows the



(a) SportsTables (b) Open Data (c) GitTables

Figure 15: Execution time breakdown of Fainder Exact compared to baselines for 999 queries.

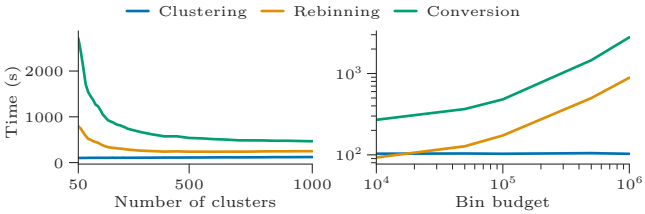


Figure 16: Construction time of Fainder on GitTables with a bin budget of 50 000 (left) and 100 clusters (right).

runtime of its three phases compared to our exact baselines. Since `profile-scan` and `binsort` both yield exact results, we can use either for the third stage of Fainder Exact. Conceptually, Fainder Exact has no guaranteed speedup as the improvement depends on the combined pruning factor of the first two stages. Nevertheless, in practice, it is 10–25× faster than `profile-scan` and 5–7× faster than `binsort`, as it can prune (a) 98%, (b) 93%, and (c) 98% of the histograms on average for our benchmark queries. Noticeably, the third iterative stage still dominates the runtime in many cases.

Note that while `binsort` appears like an overall better baseline, its performance depends not only on the number of histograms in a collection but also on the number of bins per histogram, as it creates an index entry for each bin. However, we cannot control the number of bins per histogram in our setting since they are created by the data owners. In a side experiment, we verified that the runtime of `binsort` increases linearly if we simulate collections with more average bins per histogram. Therefore, the optimal algorithm choice for stage three depends on the histogram collection, while both choices generally yield speedups over the baselines.

Index Construction. In Figure 16, we measure the index construction time on GitTables and vary the number of clusters k while keeping the bin budget $\mathcal{B}$ fixed and vice versa. We divide index construction time into clustering and histogram alignment, separating rebinning and conversion for histogram alignment. We observe that the impact of k and $\mathcal{B}$ on clustering time is negligible, as k-Means has robust scalability and Fainder's aligned bins are only assigned after the histograms have been clustered. Rebinning and conversion time decreases with an increasing k because the index becomes smaller with more evenly distributed clusters, resulting in fewer percentiles to compute. Conversely, $\mathcal{B}$ increases histogram alignment time by increasing the number of percentiles in the index. Overall, the index construction time for a reasonable selection of k is feasible, considering that histogram insertion and deletion are incremental. We only need to construct Fainder from scratch at the beginning or when processing significant bulk updates.

7.3 Solution Effectiveness

We analyze the effectiveness of Fainder in three parts: We discuss a qualitative case study of distribution-aware dataset search on existing search engines, examine the findings of our hyperparameter grid search, and finally compare Fainder's accuracy with our baselines.

Case Study. We conducted a qualitative case study to estimate the time a human needs to run a search with percentile requirements, combining keyword queries and manual investigation. For this, we continued our initial motivating example to show the current user experience's shortcomings concretely. We used Kaggle Datasets [32] in our study, as it is a comprehensive collection of datasets specifically intended for machine learning (i.e., a use case that benefits from percentile predicates). As of February 2024, our example's keyword query "lung cancer age" yielded 67 results that our data scientist needs to review. We make a conservative estimate and assume they need one minute per dataset on average, including datasets they can quickly rule out and datasets they have to download and analyze with a tool like Pandas. This would take them about one hour instead of less than a second with Fainder. Since users often search for datasets in a work context, this showcases the considerable economic potential of distribution-aware search.

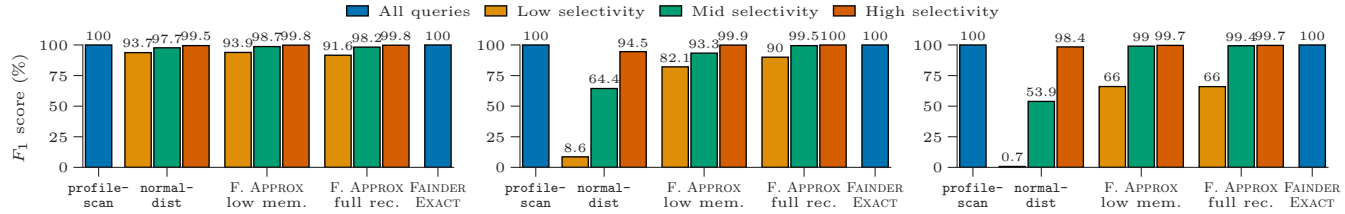
Hyperparameter Grid Search. We conducted an extensive grid search on our validation query set to analyze Fainder's configuration robustness. Below, we summarize our key insights regarding the choice of clustering algorithm (A), clustering feature transformation (T), number of clusters (k), and bin budget ($\mathcal{B}$).

(A) We examined three classes of clustering algorithms: density-based clustering with HDBSCAN [11], agglomerative clustering, and k-Means. Density-based clustering does not require choosing the hyperparameter k. However, it can classify an arbitrarily large share of points as outliers. This is detrimental in our setting since we need to index all histograms, and forming a heterogeneous outlier cluster yields catastrophic query accuracy within that cluster. Agglomerative clustering makes no assumptions about the feature distribution and is robust to outliers. Yet, it does not achieve a feasible runtime for large dataset collections; we aborted a grid search on GitTables because it took more than 2.5 days, while k-Means took less than 5 hours for the same search. k-Means achieved the best overall performance, producing clusterings that yield accurate indices within less than two minutes for each dataset collection.

(T) Next to a quantile transform [49], we also investigated simple and robust standardization, as well as no feature preprocessing in our experiments. If the features fall into clearly separable clusters by default, such as with SportsTables, we found that simple standardization or no preprocessing can yield the best results. However, across dataset collections and index configurations, the quantile transform is most robust to heterogeneous histogram features and thus produces the smallest and most accurate indices on average.

(k) Using a reasonably large value for k (> 100) generally yields a good result accuracy. Increasing k beyond this point presents a trade-off between index size and runtime, as k has a linear impact on the runtime but reduces the number of computed percentiles.

($\mathcal{B}$) The bin budget should be chosen according to the memory capacity and in conjunction with k, as more clusters result in smaller indices. A higher bin budget generally produces more precise indices, although with diminishing returns.



(a) SportsTables (b) Open Data (c) GitTables

Figure 17: F_1 score of `profile-scan`, `normal-dist`, Fainder Approx, and Fainder Exact, grouped by query selectivity.

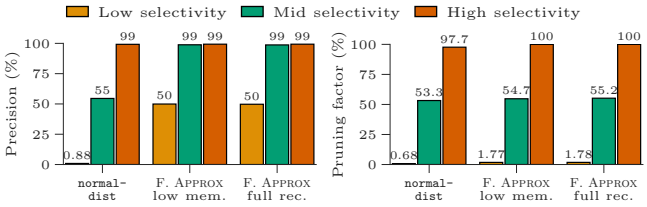
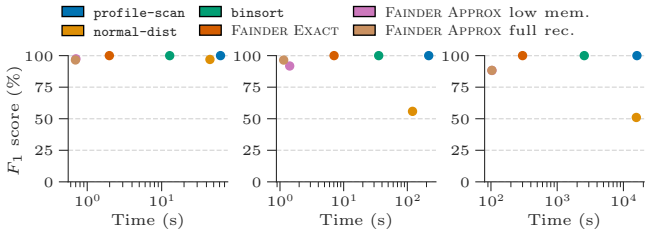


Figure 18: Precision and pruning factor of approximate solutions on GitTables.



(a) SportsTables (b) Open Data (c) GitTables

Figure 19: F_1 score over runtime of `profile-scan`, `normal-dist`, `binsort`, and Fainder variants.

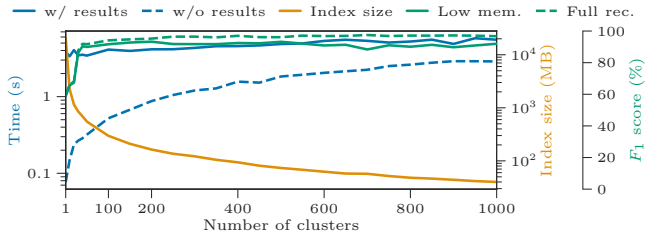


Figure 20: Query runtime, index size, and F_1 score of Fainder Approx over the number of clusters on Open Data.

Based on our grid search results, we selected index configurations that optimize result accuracy and reduce the index size for our other experiments. Concretely, we use k-Means and preprocess all collections but SportsTables using a quantile transform. In addition, we use (230, 250, 750) clusters and (5K, 50K, 100K) bins for SportsTables, Open Data, and GitTables.

Accuracy Comparison. Figure 17 presents the F_1 score of `profile-scan`, `normal-dist`, Fainder Approx low memory (based on rebinning), Fainder Approx full recall (based on conversion), as well as Fainder Exact. For the approximate solutions, we group the results by query selectivity. We observe that all approaches perform best on SportsTables, with its manually curated datasets that often match a normal distribution. For Open Data and GitTables, the results are more diverse. `normal-dist` performs consistently worse than Fainder Approx variants, especially on the more challenging collections. Fainder Approx constructed with conversion performs better than rebinning on Open Data due to its guaranteed recall of 1 but at the cost of using more memory. Overall, low and medium selectivity queries have a lower F_1 score.

To explain the worse F_1 score for queries with lower selectivity, we show the precision and pruning factor of approximate solutions on GitTables in Figure 18. Comparing the results to the F_1 score from Figure 17 (c), we observe that a lower precision predominantly causes the performance decrease. Considering the pruning factor, we see that Fainder Approx variants nevertheless filter out around 98% of the histograms. If the cardinality of the true result is low, a small absolute number of false positives can cause a sizeable relative performance decrease in precision. Thus, we argue that the worse performance for low selectivity queries has limited practical impact, as the absolute number of false results is small.

We also jointly investigated each approach's result accuracy and runtime performance. Figure 19 highlights that Fainder strictly dominates the baselines in a skyline analysis for all three dataset collections. Fainder Exact achieves 100% F_1 score in 5–53× less time than the exact baselines. Fainder Approx variants achieve the fastest runtime overall at the cost of producing a few false results. `normal-dist` presents a worse accuracy-runtime trade-off than our solutions on all collections. Therefore, we argue that its space savings ($O(2)$ vs. $O(\mathcal{B}_c)$ per column) are not justified unless the vast majority of columns follow a normal distribution.

7.4 Micro-Benchmarks

We conclude our experimental evaluation with a holistic analysis of the impact of the number of clusters and the bin budget on all three dimensions of Fainder's performance. Finally, we distill our investigation into two easy-to-follow steps for practitioners.

Impact of Clustering. Figure 20 demonstrates the interplay of runtime, result accuracy, and index size for a varying number of clusters and 50 000 bins on Open Data. Without a sufficient number of clusters ($k < 10$ in this case), accuracy and index size deteriorate significantly as the aligned bins become less precise and the clusters less balanced. For $k = 1$, the memory consumption (and index construction time) becomes unfeasible. Thus, clustering is a critical component of Fainder. Fortunately, the choice of k for larger values is robust concerning the result accuracy, making Fainder easy to configure. This experiment supports our grid search's finding that increasing k presents a trade-off between index size and runtime. However, when comparing the two blue lines, we see that the runtime impact of a growing k is partly mitigated if we also consider the result processing time. This is because more clusters can yield a more precise index, resulting in a smaller result set size.

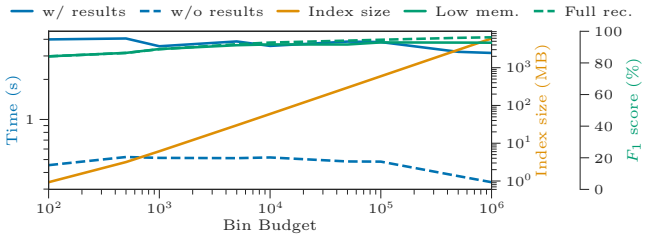


Figure 21: Query runtime, index size, and F_1 score of FAINDER APPROX over varying bin budgets on Open Data.

Impact of Bin Budget. In Figure 21, we fix the number of clusters to 100 and vary the bin budget. We observe that (1) the runtime is robust to the bin budget due to our use of binary search; (2) the F_1 score increases from 84% to 93% and 96% with an increasing bin budget, although at diminishing returns; and (3) the index size grows linearly with $\mathcal{B}$. Note that, in general, the size of FAINDER scales with the number but not the size of the datasets. This makes FAINDER relatively large for dataset collections with many small datasets, such as GitTables. On the other side, it is robust to dataset collections for machine learning, where individual datasets can take up many gigabytes or even terabytes of storage.

Practical Guidance for Index Configuration. First, select the largest possible value of k acceptable for runtime. To ensure performance, k should be at least 100× smaller than $|\mathcal{H}|$. Then, choose the largest possible value of $\mathcal{B}$ acceptable in terms of index size.

8 RELATED WORK

Prior work on dataset search can be broadly classified into data lake navigation [42, 43, 47], data discovery by example [50], task-driven search [23] for machine learning pipelines [6], and query-driven search. We focus our discussion on the latter, as the other three concepts require access to a dataset collection's raw data. Within query-driven dataset search, we distinguish two central settings: data lakes, where the raw data are fully accessible, and metadata-based search, where search engines must rely on data owner-provided, heterogeneous, and potentially missing metadata. Moreover, there are different types of search predicates. In the following, we specifically discuss distribution-aware and keyword search. We close by reviewing related work on data profiling and histograms.

Full Data Access. There is a long line of prior work on data discovery in data lakes [8, 9, 13, 21, 25, 34, 39, 60, 62, 24]. However, data lake-focused discovery techniques assume full data access and thus are not applicable in our problem setting.

Metadata Access. Existing metadata-based dataset search engines, such as Google Dataset Search [45] and CKAN [46], are limited to keyword search (possibly extended with facets). Auctus [12] is a hybrid between the full and metadata access settings, as it collects existing metadata from multiple federated data repositories but also downloads raw data from openly available datasets to profile them individually. Based on this auxiliary information, Auctus allows users to add advanced facets to their queries, such as spatiotemporal or column type filters. In general, dataset search on metadata has not yet been explored as thoroughly as data lake search. The continuum between the two settings, where more or less metadata and samples from a dataset are available, is an exciting direction for future work. In practice, solutions must deal with varying degrees of dataset profile heterogeneity and information granularity.

Distribution-Aware Search. Recently, Asudeh and Nargesian [3] developed a system vision for the new field of distribution-aware data discovery. While their vision targets the data lake setting, it highlights the importance of distributional queries over dataset collections. For the same setting, Nargesian et al. [41] proposed distribution tailoring to meet fairness requirements in data integration and Chai et al. [14] presented distribution-aware data augmentation for model training. Our work expands distribution-aware dataset search and proposes solutions that do not require raw data access.

Keyword Search. The extensive work on keyword search is surveyed in [36, 48, 58, 59]. Our contributions complement keyword search, as our query model incorporates keyword predicates and integrates them with distribution-aware (i.e., percentile) predicates. Recent advances in large language models have led to a new line of work focusing on retrieving relevant datasets given a natural language query [28, 55, 56]. However, these works either focus on textual information about datasets or require full data access, whereas we target distribution-aware queries on heterogeneous synopses.

Data Profiling. Prior work on data profiling [2] and cleaning [1, 27, 22] is orthogonal to our contributions on searching over heterogeneous histograms. There are various methods for creating histograms [16], such as equi-height or equi-width, but the problem of optimizing histograms to reduce estimation error [5, 30] or improve visibility is complementary to our setting, where data owners generate histograms independently of the search engine.

9 CONCLUSION AND FUTURE WORK

We introduced the novel problem of distribution-aware dataset search on decentralized data repositories and proposed FAINDER, a fast and accurate index for percentile predicates. FAINDER offers exact and approximate solutions, achieving more than two orders of magnitude speedups over baseline approaches and dominating the start-of-the-art in an accuracy-runtime skyline analysis.

Our work gives rise to several interesting directions for future work, of which we discuss three that we consider most important. To effectively integrate FAINDER into a system, a search engine should combine it with semantic keyword similarity search and dataset profile alignment techniques to amplify the column identifier in a percentile predicate. Once a dataset search engine supports multiple predicate types, query planning and optimization are paramount problems. Beyond system integration, we see an opportunity to leverage our query model for designing distribution-aware dataset search predicates based on data synopses other than histograms, such as sketches. For example, using synopses to rank each dataset's similarity to an input dataset or a reference distribution is challenging. Lastly, while FAINDER can approximate percentile predicates with two-sided ranges via a combination of one-sided predicates, developing an index that natively supports range predicates presents novel research challenges.

ACKNOWLEDGMENTS

We gratefully acknowledge funding from the German Federal Ministry of Education and Research under the grants BIFOLD24B and 01IS17052 (for the Software Campus project FDaaS).

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, et al. 2016. Detecting Data Errors: Where Are We and What Needs To Be Done? *Proceedings of the VLDB Endowment*, 9, 12, 993–1004.

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling Relational Data: A Survey. *The VLDB Journal*, 24, 4, 557–581.

[3] Abolfazl Asudeh and Fatemeh Nargesian. 2022. Towards Distribution-aware Query Answering in Data Markets. *Proceedings of the VLDB Endowment*, 15, 11, 3137–3144.

[4] Santiago Andrés Azcoitia and Nikolaos Laoutaris. 2022. A Survey of Data Marketplaces and Their Business Models. *SIGMOD Record*, 51, 3, 18–29.

[5] Rachel Behar and Sara Cohen. 2020. Optimal Histograms with Outliers. EDBT.

[6] Lennart Behme, Saravanan Thirumuruganathan, Alireza Rezaei Mahdiraji, et al. 2023. The Art of Losing to Win: Using Lossy Image Compression to Improve Data Loading in Deep Learning Pipelines. ICDE.

[7] Jelke Bethlehem. 2010. Selection Bias in Web Surveys. *International Statistical Review*, 78, 2, 161–188.

[8] Sagar Bharadwaj, Praveen Gupta, Ranjita Bhagwan, and Saikat Guha. 2021. Discovering Related Data at Scale. *Proceedings of the VLDB Endowment*, 14, 8, 1392–1400.

[9] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. ICDE.

[10] Arnaud Braud, Gaël Fromentoux, Benoit Radier, and Olivier Le Grand. 2021. The Road to European Digital Sovereignty with Gaia-X and IDSA. *IEEE Network*, 35, 2, 4–5.

[11] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. 2013. Density-Based Clustering Based on Hierarchical Density Estimates. PAKDD.

[12] Sonia Castelo, Rémi Rampin, Aécio Santos, et al. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proceedings of the VLDB Endowment*, 14, 12, 2791–2794.

[13] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, et al. 2018. Aurum: A Data Discovery System. ICDE.

[14] Chengliang Chai, Jiabin Liu, Nan Tang, et al. 2022. Selective data acquisition in the wild for model charging. *Proceedings of the VLDB Endowment*, 15, 7, 1466–1478.

[15] Adriane Chapman, Elena Simperl, Laura Koesten, et al. 2020. Dataset Search: A Survey. *The VLDB Journal*, 29, 1, 251–272.

[16] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. 2011. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends® in Databases*, 4, 1, 1–294.

[17] Aron Culotta. 2014. Reducing Sampling Bias in Social Media Data for County Health Inference. JSM.

[18] Jeffrey Dastin. 2018. Amazon scraps secret AI recruiting tool that showed bias against women. *Reuters*.

[19] Datarade. 2024. Find the right data, effortlessly. Retrieved Feb. 6, 2024 from https://datarade.ai/.

[20] Dawex. 2024. Data Marketplaces, Data Hubs & Data Spaces. Retrieved Feb. 6, 2024 from https://www.dawex.com/en/.

[21] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2022. MATE: Multi-Attribute Table Extraction. *Proceedings of the VLDB Endowment*, 15, 8, 1684–1696.

[22] Sainyam Galhotra, Anna Fariha, Raoni Lourenço, et al. 2022. DataPrism: Exposing Disconnect between Data and Systems. SIGMOD.

[23] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. Metam: Goal-Oriented Data Discovery. ICDE.

[24] Yue Gong, Sainyam Galhotra, and Raul Castro Fernandez. 2024. Nexus: Correlation Discovery over Collections of Spatio-Temporal Tabular Data. *Proceedings of the ACM on Management of Data*, 2, 3, 154:1–154:28.

[25] Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. 2023. Ver: View Discovery in the Wild. ICDE.

[26] Zerrin Asan Greenacre. 2016. The Importance of Selection Bias in Internet Surveys. *Open Journal of Statistics*, 6, 3, 397–404.

[27] Alon Halevy, Flip Korn, Natalya F. Noy, et al. 2016. Goods: Organizing Google's Datasets. SIGMOD.

[28] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. NAACL.

[29] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. *Proceedings of the ACM on Management of Data*, 1, 1, 30:1–30:17.

[30] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, et al. 1998. Optimal Histograms with Quality Guarantees. VLDB.

[31] Raj K. Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Computer Publishing, New York, NY.

[32] Kaggle Inc. 2024. Kaggle Datasets. Retrieved Jan. 29, 2024 from https://www.kaggle.com/datasets.

[33] Javen Kennedy, Pranav Subramaniam, Sainyam Galhotra, and Raul Castro Fernandez. 2022. Revisiting Online Data Markets in 2022. *SIGMOD Record*, 51, 3, 30–37.

[34] Christos Koutras, George Siachamis, Andra Ionescu, et al. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. ICDE.

[35] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. 2023. SportsTables: A New Corpus for Semantic Type Detection. BTW.

[36] Thuy Ngoc Le and Tok Wang Ling. 2016. Survey on Keyword Search over XML Documents. *SIGMOD Record*, 45, 3, 17–28.

[37] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28, 2, 129–137.

[38] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, MA.

[39] Renee J. Miller, Fatemeh Nargesian, Erkang Zhu, et al. 2018. Making Open Data Transparent: Data Discovery on Open Data. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 41, 2, 59–70.

[40] Molly Mulshine. 2015. A major flaw in Google's algorithm allegedly tagged two black people's faces with the word 'gorillas'. Business Insider. Retrieved Feb. 6, 2024 from https://www.businessinsider.com/google-tags-black-people-as-gorillas-2015-7.

[41] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2021. Tailoring data source distributions for fairness-aware data integration. *Proceedings of the VLDB Endowment*, 14, 11, 2519–2532.

[42] Fatemeh Nargesian, Ken Pu, Bahar Ghadiri-Bashardoost, et al. 2023. Data Lake Organization. *IEEE Transactions on Knowledge and Data Engineering*, 35, 1, 237–250.

[43] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, et al. 2020. Organizing Data Lakes for Navigation. SIGMOD.

[44] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. 2016. Automated Quality Assessment of Metadata across Open Data Portals. *Journal of Data and Information Quality*, 8, 1, 1–29.

[45] Natasha Noy, Matthew Burgess, and Dan Brickley. 2019. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. WWW.

[46] Open Knowledge Foundation. 2022. CKAN. Retrieved Aug. 28, 2022 from http://ckan.org/.

[47] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, et al. 2021. RONIN: Data Lake Exploration. *Proceedings of the VLDB Endowment*, 14, 12, 2863–2866.

[48] Jaehui Park and Sang-goo Lee. 2011. Keyword Search in Relational Databases. *Knowledge and Information Systems*, 26, 2, 175–193.

[49] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 85, 2825–2830.

[50] El Kindi Rezig, Anshul Bhandari, Anna Fariha, et al. 2021. DICE: Data Discovery by Example. *Proceedings of the VLDB Endowment*, 14, 12, 2819–2822.

[51] Adam Rose. 2010. Are Face-Detection Cameras Racist? *Time*.

[52] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A Sketch-based Index for Correlated Dataset Search. ICDE.

[53] Tess Townsend. 2017. Most engineers are white — and so are the faces they use to train software. Vox. Retrieved Feb. 6, 2024 from https://www.vox.com/2017/1/18/14304964/data-facial-recognition-trouble-recognizing-black-white-faces-diversity.

[54] Jonas Traub, Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2021. Agora: Bringing Together Datasets, Algorithms, Models and More in a Unified Ecosystem. *SIGMOD Record*, 49, 4, 6–11.

[55] Fei Wang, Kexuan Sun, Muhao Chen, et al. 2021. Retrieving Complex Tables with Multi-Granular Graph Representation Learning. SIGIR.

[56] Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data Discovery Using Natural Language Questions Via A Self-Supervised Approach. *Proceedings of the ACM on Management of Data*, 1, 4, 262:1–262:27.

[57] Bifan Wei, Jun Liu, Qinghua Zheng, et al. 2013. A Survey of Faceted Search. *Journal of Web Engineering*, 12, 1, 41–64.

[58] Jianye Yang, Wu Yao, and Wenjie Zhang. 2021. Keyword Search on Large Graphs: A Survey. *Data Science and Engineering*, 6, 2, 142–162.

[59] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. 2010. Keyword Search in Relational Databases: A Survey. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 33, 1, 67–78.

[60] Haoxiang Zhang, Aécio Santos, and Juliana Freire. 2021. DSDD: Domain-Specific Dataset Discovery on the Web. CIKM.

[61] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval using Semantic Similarity. WWW.

[62] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. SIGMOD.

[63] Jiongli Zhu, Sainyam Galhotra, Nazanin Sabri, and Babak Salimi. 2023. Consistent Range Approximation for Fair Predictive Modeling. *Proceedings of the VLDB Endowment*, 16, 11, 2925–2938.