LakeCompass: An End-to-End System for Data Maintenance, Search and Analysis in Data Lakes

Chengliang Chai*
ccl@bit.edu.cn
Beijing Institute of Technology

Yuhao Deng*
dyh18@bit.edu.cn
Beijing Institute of Technology

Yutong Zhan
zyt@bit.edu.cn
Beijing Institute of Technology

Ziqi Cao
caoziqi@bit.edu.cn
Beijing Institute of Technology

Yuanfang Zhang
zyuanfang@bit.edu.cn
Beijing Institute of Technology

Lei Cao
lcao@csail.mit.edu
University of Arizona/MIT

Yuping Wang
wyp_cs@bit.edu.cn
Beijing Institute of Technology

Zhiwei Zhang
zwzhang@bit.edu.cn
Beijing Institute of Technology

Ye Yuan
yuan-ye@bit.edu.cn
Beijing Institute of Technology

Guoren Wang
wanggr@bit.edu.cn
Beijing Institute of Technology

Nan Tang
nantang@hkust-gz.edu.cn
HKUST (GZ)

ABSTRACT

Searching tables from poorly maintained data lakes has long been recognized as a formidable challenge in the realm of data management. There are three pivotal tasks: keyword-based, joinable and unionable table search, which form the backbone of tasks that aim to make sense of diverse datasets, such as machine learning. In this demo, we propose LakeCompass, an end-to-end prototype system that maintains abundant tabular data, supports all above search tasks with high efficacy, and well serves downstream ML modeling. To be specific, LakeCompass manages numerous real tables over which diverse types of indexes are built to support efficient search based on different user requirements. Particularly, LakeCompass could automatically integrate these discovered tables to improve the downstream model performance in an iterative approach. Finally, we provide both Python APIs and Web interface to facilitate flexible user interaction.

PVLDB Reference Format:
Chengliang Chai, Yuhao Deng, Yutong Zhan, Ziqi Cao, Yuanfang Zhang, Lei Cao, Yuping Wang, Zhiwei Zhang, Ye Yuan, Guoren Wang, and Nan Tang. LakeCompass: An End-to-End System for Data Maintenance, Search and Analysis in Data Lakes . PVLDB, 17(12): 4381 - 4384, 2024.
doi:10.14778/3685800.3685880

1 INTRODUCTION

Many organizations have acknowledged the importance of providing open access to their tabular data for the public good [14]. However, the large amount of tables are often poorly maintained (*e.g.*, lacking of index or schema information). To make sense of this data, a key initial step for the data scientists is to search tables relevant to the task at their hand. To this end, we propose to build an end-to-end prototype system, LakeCompass that driven by the need of the downstream analytics tasks, offers efficient and effective table search over data lakes.

At a high level, LakeCompass features the following three key functionalities.

Index building. We build various types of indexes over tables in data lakes, such as inverted index, local sensitive hash (LSH) and the hierarchical navigable small world (HNSW), which taking both text similarities and semantic similarities into consideration, efficiently and effectively support table search.

Table search. We support three typical categories of table search, *i.e.*, keyword-based, joinable and unionable table search. The former one returns relevant tables based on the given keywords. The latter two take as input a query table, and discovering tables that can be integrated with the query table.

Data Selection for Model analysis. LakeCompass offers a data selection component that taking into consideration the specific need of the downstream applications, *i.e.*, the ML models, intelligently selects the tuples and features from the table search result in a more targeted manner.

Overall, LakeCompass has the following advantages.

[1. *High efficiency.*] Various types of indexes allow us to support efficient search over different queries. In terms of the model analysis module, we propose to use the online learning technique to improve the efficiency of iterative model evaluation.

[2. *High effectiveness.*] We maintain the original data associated with their semantic embeddings, together with various indexes, which allow users to search highly relevant tables customized to their needs. For the model analysis, we propose a multi-armed bandit (MAB) technique to judiciously select beneficial tuples/features among the returned relevant tables.

[3. *User-friendly interaction.*] We encapsulate the above functionalities through flexible Python APIs, where the user can upload

*Both authors contributed equally to this research.
Ye Yuan is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685880

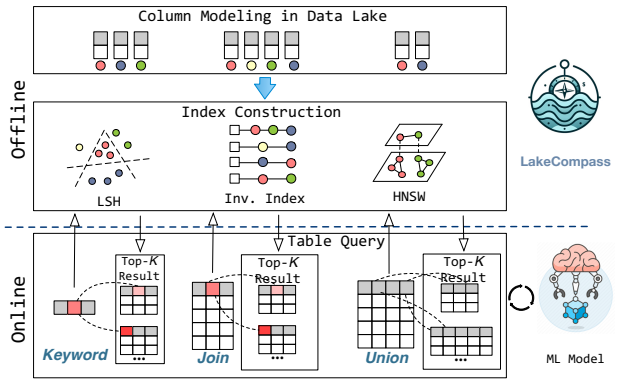

Figure 1: Overall Architecture of LakeCompass.

data, construct indexes, search tables and build ML models. We also provide a Web interface to serve table search and analysis in a more intuitive way.

Existing Works. There exist several prototypes of table search. Aurum [6] takes as input keywords and returns relevant tables based on their schema information. COCOA [5] focuses on searching joinable tables; Auctus [1] supports both joinable and unionable table search; Besides the above table search, Juneau [7] further supports feature extraction and data cleaning based on the returned tables. Similarly, DIALITE [12] can also search tables that can be integrated with the query table, and then conducts some data analysis like entity matching. Although the above prototypes supports table search in data lakes, they do not consider the semantic information of columns/rows of tables, which is rather significant in achieving high effectiveness. What's more, they do not take the machine learning task into consideration, which is the key application for table search in data lakes. Although some works [3, 9] have studied data augmentation for ML, they assume the relevant data is discovered beforehand by another system, hence not offering an end-to-end table search solution.

In summary, LakeCompass is an end-to-end system that is adequate to demonstrate the efficiency and effectiveness of table search for real-world applications.

2 SYSTEM OVERVIEW

At a high level, LakeCompass consists of the following modules: data lake embedding, index construction, online table query processing as well as the model analysis, as shown in Figure 1. The former two modules are offline, which respectively encode the column/schema in the data lake into a vector and index these vectors. Once an online query table arrives, we first encode its column(s) and then use the index to retrieve relevant tables with the highest relevance score from the data lake. If the query is a keyword, we just encode the query and retrieve relevant tables.

Embedding Data Lake. Generally speaking, column representations are significant in table search. The columns of the original tables in data lake are always represented as fixed-length vectors. The vectors are mostly hash codes [6, 17] (*e.g.*, generated by the minhash function) or embeddings [8, 10] (*e.g.*, generated by pre-trained language models) that are effective in finding columns with similar semantics or high overlaps, which are rather significant in searching joinable/unionable tables. However, some methods like Josie and InfoGather [15, 16] directly use the original cell values of each column to search highly overlapping columns rather than using vectors. Besides, we also maintain the metadata of data lake tables (*e.g.*, table name, caption and schema information, etc.) and index them, which are also important in table search, especially for keyword-based search.

Index Construction. Almost all table search methods rely on an index to search large table repositories. Building upon the representations discussed above, several typical ANN indexes like LSH [17] or HNSW [8] can be utilized to index all the embeddings. As another option, inverted index [11, 15, 16] accelerates the process of finding highly overlapping columns, where each cell value is linked to the columns that contain the value. In this case, these colorful circles represent Column IDs rather than vectors. All the types of indexes are supported in LakeCompass.

Online Table Search. For keyword-based search, the user issues the keywords, which are encoded as embeddings, and retrieve tables with similar embeddings on metadata or columns. For joinable search, LakeCompass supports users in issuing a query table through either user-uploaded files or the result of keyword-based search. Once a user issues a table T_J and a specific column c for search, LakeCompass first represents c as a vector. It then uses the index to quickly identify a of top-K columns (tables), denoted by $R(T_J)$, with high relevance scores. The unionable search follows the same procedure. The only difference is that given a table T_U, it has to search over each attribute in T_U and aggregate the results. Specifically, for each column $c_i \in T_U$, similar to the join search, we retrieve from the data lake a set of columns that have high relevance scores with c_i. Then, we compute the union of all retrieved tables. For each table T in the union set, we compute the relevance score using techniques like maximum bipartite graph matching, considering the column relevance between the T_U and T. Finally, a set of top-K tables, denoted by $R(T_U)$ with the highest relevance scores are returned. In LakeCompass, we have implemented various joinable [15–17] and unionable search [8, 11, 15] algorithms.

Iterative Data Selection for Model Analysis. When the query table serves as the train data, due to the data scarcity problem, we expect to augment more tuples/features through unionable/joinable table search. However, although all the returned tables $R(T_J)/R(T_U)$ can be integrated into the query table, integrating all of them may not improve the model performance. The reason is that they come from heterogeneous sources in data lakes, some of which are in fact noisy to the model. Therefore, we apply our proposed tuple/feature augmentation techniques [2, 13] to judiciously select tuples/features (from unionable/joinable tables), iteratively integrate to the query table and evaluate the performance. Here, we summarize the high level idea [2] of selecting tuples from unionable tables.

i). *Clustering.* Given all tables in $R(T_U)$, we first merge all tuples of these tables, and then cluster these tuples based on their feature similarities. Then tuples in each cluster can be seen as homogeneous. Next, we use an iterative MAB solution to judiciously select tuples that can benefit the model from these clusters.

ii). *MAB-based data selection.* We take each cluster as a bandit arm. Each arm is assigned with an upper confidence bound (UCB) score,

```
# build index for the datalake
datalake = LakeCompass.DataLake('/datalake_demo')
keyword_index = datalake.metadata.indexing(
    index_type='HNSW', search_type='keyword',
      config=LakeCompass.HNSWconfig())
# keyword based search
candidate_table = LakeCompass.keyword_search(
    keyword_index, 'education')
```

Figure 2: A Code Segment for Index Building and Keyword-based Search of LakeCompass.

which will be updated after each iteration of tuple selection. At the beginning of each iteration, the arm with the highest UCB score will be selected, then a small batch of tuples will be sampled from the cluster and added to train. If the model performance is tested to be improved on a validation set, a reward will be assigned. At a high level, the UCB score is computed by the summation of an exploration and an exploitation score. The less frequently a cluster is picked, the higher the exploration score. The higher reward the cluster has obtained, the higher the exploitation score. Hence, a combination of the two scores used by MAB-based method can well handle the exploration-exploitation trade-off. Specifically, for each cluster C_i, at iteration k:

$$UCB(C_i, k) = R_i^k + \alpha\sqrt{2\ln n^k/(n_i^k+1)} \quad (1)$$

where α is pre-defined, n_i^k is the total number of times that C_i was assigned non-zeros scores from iterations 1 to k, and n^k is the sum of n_i^k for all clusters at iteration k. In Eq. 1, the former part (R_i^k) refers to the *exploitation*, which means to focus more on the cluster where some tuples have resulted in much performance improvement. The latter part ($\sqrt{2\ln n^k/(n_i^k+1)}$) refers to the *exploration*, *i.e.*, focusing more on the clusters that are rarely picked.

iii). *Iterative model evaluation.* We iteratively apply the MAB-based solution until the model performance does not increase within several successive iterations. In order to accelerate the iterative training, we can apply the online learning technique that only trains over the added tuples.

Similarly, the MAB-based solution can also be extended to augment useful features from joinable tables [13].

3 DEMONSTRATION SCENARIOS

This section showcases our demonstration scenarios, which highlight the main functionalities of LakeCompass. These scenarios illustrate how LakeCompass identifies relevant tables within the data lake to enrich tuples or features, thereby enhancing the performance of downstream models. Users can utilize LakeCompass functionalities via Python APIs or Web interfaces.

Datasets. We support index construction, table search and model analysis over 1TB data collected in our benchmark [4] – **100X** larger than the data lakes used in existing works, and the number of columns is up to **100 million**, on which we implemented and evaluated multiple typical table union/join search methods. Moreover, we also provided a sufficient number of labeled diverse

```
# train configuration
predict_col = 'Educational Category'
model_config = {'model': 'SVM', 'type': 'classification',
                'k': 4, 'model_config': SVMconfig()}
query_table = LakeCompass.read('education_Des_Moines.csv')
query_table_model = DownstreamModel(**model_config)
val_set = pd.read_csv('val_set.csv')
test_set = pd.read_csv('test_set.csv')
# trian a model using query table
query_table_model.train(query_table, predict_col, val_set)
query_table_model.eval(test_set, predict_col)
```

Figure 3: A Code Segment for Training Model based on the Query Table.

```
# unionable table search
union_index = datalake.columns.indexing(
    index_type='HNSW', search_type='union',
    config=LakeCompass.HNSWconfig())
unionable_tables = LakeCompass.union_search(
    union_index, query_table)
```

Figure 4: A Code Segment for Unionable Table Search of LakeCompass

queries (more than 10 thousand) – **10X** more than any other works, so as to test the algorithm performance. To this end, we organized a team of 25 graduate students and spent more than *7,500 human hours* on labeling the queries.

(1) **Index Building.** LakeCompass enables the utilization of various index types (including the LSH, HNSW, and inverted index.) to support the typical three table search tasks. As shown in Figure 2, the user builds the HNSW index over the metadata to support keyword-based search. Note that we also support the users to build their own data lakes and build indexes.

(2) **Keyword-based Search.** Users could obtain the training dataset by keyword search. To be specific, as shown in Figure 2, The user could use the Python APIs to set "education" as a keyword and utilize the "keyword_search" function to retrieve relevant tables. Then, tables related to education in different cities of different states in the U.S. are retrieved from our data lake.

(3) **Unionable Table Search.** As shown in Figure 3, users have the flexibility to either upload a query table or choose a table from the keyword-based search as the query table. In such cases, the model based on the solely query table tends to show poor performance. Therefore, they can then employ the "union_search" function to retrieve unionable tables based on semantic similarities. As depicted in Figure 4, users select the "education_Des_Moines.csv" as the query table. Subsequently, additional tables pertaining to education in other cities within the state of Iowa are retrieved through the union search. These retrieved tables share multiple semantically identical columns with the query table, enabling them to be seamlessly unioned together.

(4) **Joinable Table Search.** Users could also interact with the Web

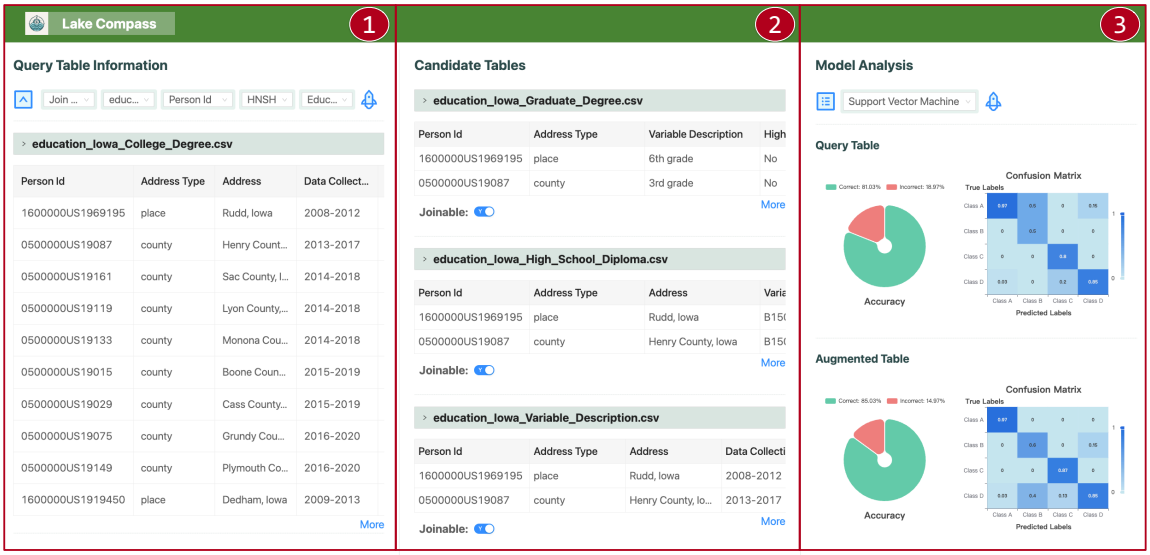

Figure 5: The Web Interface of LakeCompass

```
#  train another model using unionable tables
augmented_table_model = DownstreamModel(**model_config)
augmented_table_model = unionable_tables.auto_augment(
    augmented_table_model, val_set, predict_col)
# evaluate models
augmented_table_model.eval(test_set, predict_col)
```

Figure 6: A Code Segment for Model Analysis of LakeCompass

interface to conduct table search. For joinable search, in Figure 5 - ①, users could specify the query table, query column, label column to be predicted and the index type. Retrieved tables could be displayed in Figure 5 - ②, allowing users to eliminate or keep tables manually to improve the model performance. LakeCompass then can select data from the kept ones to improve the model.

(4) **Data Selection for Model Analysis.** LakeCompass provides an "auto_augment" function, which employs the MAB-based strategy to intelligently select the most beneficial tuples/features for the downstream model and returns the trained model. To be specific, as shown in Figure 6, the users could train the same models with identical architectures and hyperparameters based on the set of auto-augment unionable tables. This step could also be done by users through interacting with the Web interface. As shown in Figure 5 - ③, LakeCompass presents the model performance using visual aids, including a confusion matrix and an accuracy graph. The results demonstrate the model trained on the table after augmentation outperforms the model trained on the query table.

4 CONCLUSION

In this paper, we present LakeCompass, an end-to-end prototype system offering index building, table search, and model analysis. LakeCompass enhances the performance of downstream machine learning tasks by iteratively integrating relevant tables. It offers a user-friendly Web interface for flexible user interaction.

ACKNOWLEDGMENTS

This paper is supported by the NSFC (62102215, 62072035, 61932004, 62225203, U21A20516, U23A20297, U2001211, U23B2019), CCF-Huawei Populus Grove Fund (CCF-HuaweiDB202306), the National Key R&D Program of China(2022YFB2702100,2021YFB2700700), the DITDP (JCKY2021211B017), and the Fundamental Research Funds for the Central Universities. Lei Cao is supported by the NSF (DBI-2327954) and Amazon Research Award.

REFERENCES

[1] Sonia Castelo and Rémi Rampin et al. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proc. VLDB Endow.* 14, 12 (2021), 2791–2794.
[2] Chengliang Chai and Jiabin Liu et al. 2022. Selective Data Acquisition in the Wild for Model Charging. *Proc. VLDB Endow.* 15, 7 (2022), 1466–1478.
[3] Nadiia Chepurko and Ryan Marcus et al. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proc. VLDB Endow.* (2020).
[4] Yuhao Deng, Chengliang Chai, and Lei Cao et al. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. In *VLDB 2024.*
[5] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In *EDBT 2021.*
[6] Raul Castro Fernandez et al. 2018. Aurum: A Data Discovery System. In *ICDE.*
[7] Yi Zhang et al. 2019. Juneau: Data Lake Management for Jupyter. *VLDB.* (2019).
[8] Grace Fan and Jin Wang et al. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *PVLDB* (2023).
[9] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. Metam: Goal-Oriented Data Discovery. In *ICDE 2023.*
[10] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *NAACL-HLT 2021.*
[11] Aamod Khatiwada, Grace Fan, and Roee Shraga et al. 2022. SANTOS: Relationship-based Semantic Table Union Search. *CoRR* abs/2209.13589 (2022).
[12] Aamod Khatiwada, Roee Shraga, and Renée J. Miller. 2023. DIALITE: Discover, Align and Integrate Open Data Tables. In *SIGMOD/PODS 2023.*
[13] Jiabin Liu, Chengliang Chai, and Yuyu Luo et al. 2022. Feature Augmentation with Reinforcement Learning. In *ICDE 2022.*
[14] Renée J. Miller and Fatemeh Nargesian et al. 2018. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.* 41, 2 (2018), 59–70.
[15] Mohamed Yakout and Kris Ganjam et al. 2012. InfoGather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD 2012.*
[16] Erkang Zhu and Dong Deng et al. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD 2019.*
[17] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016).