SET: Searching Effective Supervised Learning Augmentations in Large Tabular Data Repositories

ABSTRACT

Successful supervised learning models rely on predictive features, which rarely come from a single dataset. As a result, relevant datasets need to be integrated before training the actual model. This raises one natural question: *"how can one efficiently search for predictive features from relevant datasets for integration with responsible AI guarantees?"*. This paper formalizes the question as the *data augmentation search problem* with an objective of minimizing the search latency. We propose SET, an interactive system that intakes a supervised learning task and searches for a set of join-compatible datasets that optimally improve the performance of the task. Specifically, SET manages a corpus of relational datasets, uses linear regression as a *proxy model* to evaluate augmentation candidates, and applies *factorized machine learning* to accelerate model training and evaluation algorithmically. Furthermore, SET leverages system and hardware optimizations to maximize parallelism across augmentation searches. These allow SET to search for a good augmentation plan over 1 million datasets with a latency of 1.4 seconds.

ACM Reference Format:

. 2024. SET: Searching Effective Supervised Learning Augmentations in Large Tabular Data Repositories. In *Governance, Understanding and Integration of Data for Effective and Responsible AI (GUIDE-AI '24), June 14, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665601.3669847

1 INTRODUCTION

Existing relational data repositories [4, 15] present themselves as potential data sources to improve supervised learning tasks. However, the conventional method of keyword-based searching through dataset metadata [5] requires significant manual efforts on integration and evaluation. As a result, task-based data search has emerged as an autonomous solution for finding relevant datasets to improve data-centric tasks such as supervised learning [7, 9, 10, 12–14, 16]. These search systems intake a task represented by a dataset T and a target variable Y, and identify datasets to augment the dataset by joining with T to add new features (*vertical augmentation*). Training an SL model on this augmented data can significantly improve its accuracy. To maximize the accuracy of a downstream SL task, the search system aims to find the best sequence of vertical augmentations (*augmentation plan*). To achieve responsible AI, human-in-the-loop is often necessary in deciding an integration for datasets discovered by the search system.

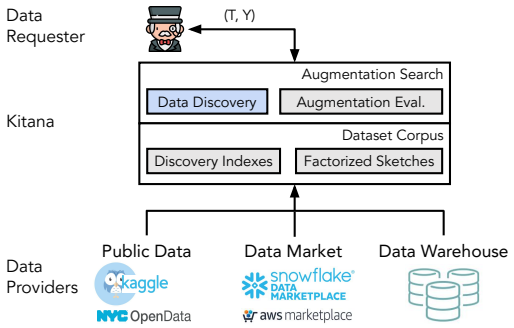

Figure 1: Overview of SET Architecture.

For human-in-the-loop tasks, latency is a crucial criteria; even in a large data repository with over 1M join compatible features, human expect quick and interactive responses (~1s) to maintain engagement. This paper formalizes the data preparation step for a supervised learning task as the *data augmentation search problem* where the objective is to minimize latency in identifying relevant datasets. However, maintaining low latency is challenging for data augmentation searching systems when processing large data repositories due to expensive evaluation and the exponential search space. Existing works identify a set of candidate tables for augmentation and evaluate the prediction accuracy of each augmentation. For example, techniques like ARDA [7] support *vertical augmentation*; it materializes the augmented dataset and retrains a model to assess the augmentation quality. Targeting general data tasks, Metam [10] proposes clustering similar datasets to reduce the number of augmentation candidates and achieves comparable accuracy enhancements. The main limitation of these approaches is that they need to materialize joins, followed by time-consuming retraining, making it impossible to achieve interactive speeds.

Saibot [12] is a differentially private data augmentation search platform that leverages concepts in factorized learning to avoid materializing joins. Firstly, Saibot introduces a cheap-to-train *proxy model*, linear regression, to assess the quality of the prediction task over augmentation candidates. Secondly, Saibot greedily finds the next optimal *vertical augmentation* candidate, with a non-trivial accuracy improvement, to account for the target variable Y; this approach reduces potential false positives over augmentation candidates. Finally, Saibot incorporates semi-ring aggregations to enhance its performance. Several semi-ring structures have been designed to represent common statistics, such as counts and sums, which support training a wide range of machine learning models, including linear regression. One of the key strengths of these semi-ring structures is their inherent distributive property. This

GUIDE-AI '24, June 14, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0694-3/24/06…$15.00
https://doi.org/10.1145/3665601.3669847

property allows for the efficient computation of statistics across joins, even without the expensive materialization. As a result, Saibot pre-computes semi-ring structures for each dataset differential privately, indexing these structures by join keys to enhance data discovery. This approach allows Saibot to achieve SOTA performance in evaluating each augmentation candidate (~1-5ms), whereas existing data search systems [7, 10] take seconds. However, even though it is cheap to evaluate each augmentation candidate, sequentially evaluating all of them in a large data corpus (>1M join compatible features) is still expensive, even with multi-threading. This is because Saibot has not fully harnessed hardware optimizations.

One key observation is that training and evaluating augmentation candidates can be rewritten as matrix operations over the semi-ring structures between matching join keys, making GPU a good fit for optimization. However, a direct application of GPUs faces a challenge due to their need for substantial data sizes and uniform operations to operate effectively: first, each augmentation candidate is too small to fully utilize GPU capacity, and second, different augmentation candidates join differently with T. As a result, even though naively clustering datasets solves the data size problem, the second challenge still persists. To allow fast evaluation of all augmentation candidates in the data corpus while guaranteeing a comparable quality of the augmentation plan, we propose SET, a GPU-based data augmentation search system that builds on top of Saibot. SET addresses the challenges by proposing a novel search algorithm that (1) pre-clusters augmentation candidates based on join key domains and (2) evaluates augmentation candidates on the residuals of target variable Y, instead of Y itself. This approach transforms the task of evaluating each augmentation candidate into simultaneously evaluating a cluster of augmentation candidates. Within each cluster, evaluating each augmentation candidate becomes independent vector multiplications of the same dimensionality, making it possible to leverage GPU parallelization to expedite the search process and ensure interactive latency.

SET's key result is depicted in Figure 2, highlighting accuracy improvement over time. This was tested on a collection of 322 datasets from NYC Open Data [4], 741 datasets from Data.gov [8], and a synthetic dataset containing 1M join compatible features. With clustering by join key and residual fitting, SET greatly reduces the search latency between iterations from ~ 150s to ~ 1s. The overall quality, measured in terms of accuracy, of the finalized *augmentation plan* is consistent with that of Saibot.

2 BACKGROUND

Problem Definition. Let $\mathcal{D} = \{D_1, D_2, \dots\}$ be a relational data repository where each D_i is provided by some data provider. A user issues a supervised learning task by providing a dataset T that contains the target variable Y. During the augmentation search, SET aims to find a good *augmentation plan* $\mathcal{P} = [A_1, \cdots, A_n]$ that specifies a sequence of vertical augmentation A: $A(I)$ applies an augmentation to a dataset I, and is defined as $A(I) = I \bowtie_{j_A} D_A$ for vertical augmentation[1]. Applying augmentation plan to the training dataset T is defined as $\mathcal{P}(T) = A_n(A_{n-1}(\cdots A_1(T)))$. More formally, the augmentation search problem can be defined as

[1]For notational convenience, we assume the j_A remains consistent between I and D_A. More generally, SET employs Aurum [9] to discover augmentation candidates, which supports equijoins; in fact, the factorized learning techniques support any type of join, including theta and anti-joins.

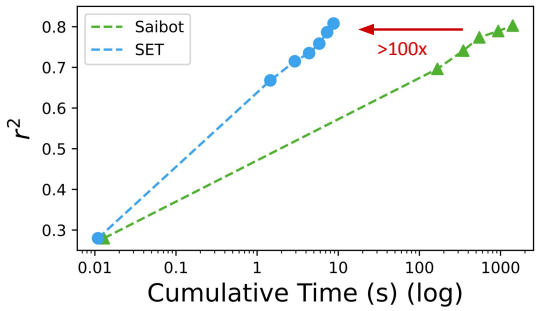

Figure 2: The accuracy of the data augmentation discovered by SET for the SAT dataset from a corpus with > 1M join-compatible features. The accuracy in the y-axis uses the r^2 score of a linear regression model evaluated on the discovered *augmentation plan*; the x-axis measures the latency in seconds with the logarithm scale.

Problem 1 (Task-Based Data Search.). *Given a dataset T and a target variable Y as input where T, find an augmentation plan $\mathcal{P}^*$ that maximizes test accuracy with respect to T:*

$$\mathcal{P}^* = \operatorname*{argmax}_{\mathcal{P}} acc(\theta, \mathcal{P}(T))$$
$$s.t. \quad \theta = M.train(\mathcal{P}(T)),$$
$$\theta \in M \qquad \textit{(model type constraint)}$$

System Overview. Figure 1 presents the system components and control flow, with components enclosed in black boxes representing integral parts of SET, and the component in blue can be outsourced to an external service (we use Aurum for augmentation candidate discovery). At a high level, SET's components are divided into **online** and **offline** phases. While offline, SET collects a large volume of datasets as augmentation candidates, preprocesses them, and builds indexes to serve requests efficiently in the online phase. During the online phase, SET greedily constructs the optimal augmentation plan $\mathcal{P}^*$ in Problem 1.

Offline Phase. SET clusters datasets in $\mathcal{D}$ sharing similar join keys, forming distinct clusters $\{C_1, C_2, \dots\}$. For each cluster C_i, a join key domain $\mathcal{J}_i$ is specified (detailed in Section 3). Additionally, SET computes sketches to facilitate data discovery and augmentation search—it pre-aggregates attribute profiles with respect to each dataset's corresponding join key domain. SET also implements basic cleaning, standardization, and feature transformations.

Online Phase. The user submits request (T, M), SET sets the initial $\mathcal{P}^*$ to be an empty set, and finds *horizontal augmentation* candidates by matching schemas and *vertical augmentation* candidates by matching join key domains. For each iteration, SET greedily adds the best augmentation candidate to the current augmentation plan. This process continues if the improvement on accuracy, (adjusted) r^2 score in our implementation, of the *proxy model* evaluated on $\mathcal{P}(T)$ is over a hyper-parameter δ.

Annotated Relations and Semi-ring Aggregates. The annotated relational model, as described in [6], maps a tuple $t \in D_i$ to a commutative semi-ring $(\mathbb{D}, \oplus, \otimes, 0, 1)$, where $\mathbb{D}$ is a set, $\oplus$ and $\otimes$ are commutative binary operators closed over $\mathbb{D}$. The elements

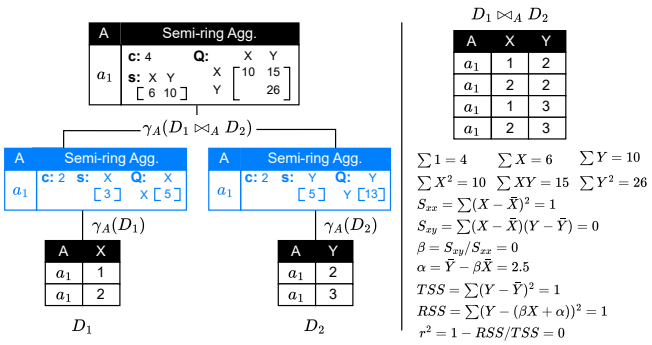

Figure 3: Factorized learning optimization to compute sufficient statistics for simple linear regression over $D_1 \bowtie_A D_2$.

0/1 represent the zero/unit elements in $\mathbb{D}$ satisfying $x \oplus 0 = x$ and $x \otimes 1 = x$. As a concrete example, ordinary addition (+) and multiplication ($\times$) over real numbers, $\mathbb{R}$, form a commutative semi-ring. An annotation for $t \in D_i$ is denoted as $D_i(t)$, and $D_i(t) = 0$ for $t \notin D_i$. Some group-by aggregations, unions and joins can be rewritten as operations over annotations:

- $(\gamma_{\mathbf{A}} D_i)(t) = \sum\{D_i(t_1) \mid t_1 \in D_i, \pi_{\mathbf{A}}(t) = \pi_{\mathbf{A}}(t_1)\}$.
- $(D_1 \bowtie D_2)(t) = D_1(\pi_{S_{D_1}}(t)) \otimes D_2(\pi_{S_{D_2}}(t))$.

(1) The annotation of tuple t after a group-by sum aggregation is the sum of all annotations sharing the same $\mathbf{A}$ with t. Here, $\sum$ represents pairwise $\oplus$ operation across all annotations.

Factorized Linear Regression. SET employ factorized learning to train linear regression over augmented datasets and prevent the expensive cost of materializing the joins and unions. Here, we provide the high-level intuition; the technical details can be found in [13]. Given the training data $(\mathbf{X}, \mathbf{y})$ for $\mathbf{X} \in \mathbb{R}^{n \times k}$ and $\mathbf{y} \in \mathbb{R}^{n \times 1}$, the analytic solution for a linear regression model is $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. If we treat Y as a special feature, the core statistics in the analytic solution are the sum of pairwise products between features ($\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$). These statistics can be be derived from aggregating annotations instead of the original data. By leveraging the fact that $\otimes$ can be distributed over $\oplus$ in commutative semi-rings, we may aggregate features locally per join key before joining the relations to train linear regression over them. Such pre-aggregation step is conducted offline. Then, aggregated semi-ring sketches can be combined using $(c_1, \mathbf{s}_1, \mathbf{Q}_1) \otimes (c_2, \mathbf{s}_2, \mathbf{Q}_2) = (c_1c_2, c_1\mathbf{s}_2 + c_2\mathbf{s}_1, c_1\mathbf{Q}_2 + c_2\mathbf{Q}_1 + \mathbf{s}_1\mathbf{s}_2^T)$. Finally, a sum aggregation ($\oplus$) over the annotations across all join keys yields $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$. We provide an example in Figure 3 to illustrate how aggregates (c, s, Q for $0^{th}, 1^{st}$, and 2^{nd} order monomials among features) can be computed via local aggregation as highlighted in blue. For concreteness, consider the statistics $\sum X = 6$ over $D_1 \bowtie D_2$ in Figure 3; this can be computed by locally summing X across join key a_1 in D_1, multiplied by the count across a_1 in D_1.

3 PERFORMANCE OPTIMIZATION

The online search algorithm for $\mathcal{P}$ is detailed in Algorithm 2. We use forward feature selection to greedily select the next most beneficial feature that enhances model accuracy. This involves training and evaluating supervised learning models on augmented datasets

Algorithm 1 Vertical Augmentation Search Algorithm

1: **Input:** T, Y
2: $\mathcal{P} \leftarrow \{\}$
3: $T[Y] \leftarrow$ update $T[Y]$ with prediction residuals
4: **for** k times **do**
5: **for all** $D_i \in \mathcal{D}$ **do**
6: **for all** $f \in D_i$ **do**
7: Augment T with f, train and evaluate
8: **end for**
9: **end for**
10: $f_{opt} \leftarrow$ feature improving accuracy the most.
11: Greedily add the next best $f_{opt} \in D_{opt}$ to $\mathcal{P}$
12: $T \leftarrow$ Augment T with D_{opt} and update residual
13: **end for**
14: **Return:** $\mathcal{P}$

Algorithm 2 Augmentation Search Algorithm

1: $T \leftarrow$ Original input dataset
2: **for all** all augmentation candidate in data corpus **do**
3: Evaluate augmentation by augmenting T with the candidate
4: **end for**
5: **Return:** Optimal augmentation candidate

for each augmentation candidate within $\mathcal{D}$ (Lines 5-9). Although this search process allows for high parallelization across augmentation candidates, scaling the process to match the cardinality of $\mathcal{D}$ is still infeasible, especially for large relational data repositories. Minimizing latency between search iterations could be significantly enhanced by clustering multiple augmentation candidates and evaluating them simultaneously within a single thread. With this objective, SET pre-clusters augmentation candidates according to their join key domains and merges semi-ring structures for all candidates within each cluster. It then uses a boosting-like method to predict the *residuals* of Y for each search iteration.

3.1 Hardware Optimization

The semi-ring annotations allow us to rewrite training linear regression over joins as matrix operations over matching join keys. This makes GPUs well-suited for accelerating such operations. However, using GPUs effectively is challenging: (1) We need a large number of threads, typically at least a thousand, to take advantage of a GPU's capabilities. (2) GPU threads are less versatile than CPU threads, particularly in handling varied tasks with branching. They require uniform, straightforward instructions for each task. In data search, different augmentation candidates join differently with the input dataset T (L6-8 in Algorithm 2), evaluating a *single* join candidate using GPUs (1) and the varying join key value sets for each augmentation candidate (2) can limit the effectiveness of GPU acceleration.

Cluster by Join Key Domains. To tackle challenge (1), GPU acceleration is selectively applied to popular join keys with a high number of join candidates, focusing primarily on spatial, temporal, and domain-specific common key types such as District Borough Number in NYC open data [4]. For challenge (2), we implement an offline preprocessing phase. This phase clusters augmentation candidates based on their join key value sets using *hierarchical*

clustering, based on the *Jaccard similarity* of the join key domain with a minimum similarity of 0.5. For each cluster, we construct three tensors in PyTorch, each of size $m \times k$, where m is the domain size and k is the number of features. These tensors correspond to $\sum f^2, \sum f, \sum 1$ for each feature f (i.e. X in Figure 3), including the special feature Y, for each join key, and are initialized with zero for absent domain values. These tensors are stored on disk and transferred to the GPU as needed. We use GPU acceleration for join keys with at least a cluster size of 1K.

3.2 System Optimization

Through clustering by join key domains, the number of threads required to parallel the search process reduced from the number of datasets in $\mathcal{D}$ to the number of clusters. We exploit system optimization to evaluate various clusters simultaneously.

- **Multiple threads.** On a single machine, we use multiple threads, with each thread dedicated to identifying the optimal feature within a specific cluster.
- **Distributed Systems.** SET distribute clusters across different machines. Each machine independently identifies the best features among those allocated to it. When a cluster exceeds the memory capacity of the GPU, SET partitions the features of the cluster into batches. All batches are indexed by the same join key domain, and distributed to different machines.

3.3 Algorithmic Optimization

While the system and hardware optimizations allows perfect parallel execution, they introduce a new challenge: for each iteration and a fixed augmentation feature f, SET needs to compute $\sum f_i f_j$ for $f_i, f_j \in \mathcal{P} \cup \{f\}$. The number of unique $\sum f_i f_j$ increases quadratically with respect to the number of features in $\mathcal{P}$. Further, the dimension of the required statistics $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$ increases linearly where computing the matrix inverse could be expensive. To align better with the GPU architecture, we improve the model training and evaluation process (L7) with the following algorithmic optimizations motivated by forward-stagewise regression [11]:

- **Residual Fitting.** Given the input dataset T, SET leverages an existing AutoML service, such as FLAML [17], and uses features in T to predict the target variable Y. It then updates Y to be the residual of this prediction (L3 in Algorithm 2). SET then re-calibrate the semi-ring structure of T, which only contains $\sum 1, \sum Y, \sum Y^2$ for each join key.
- **Boosting.** Inspired by boosting, for each iteration of the greedy search, rather than combining all features in $\mathcal{P}$ with the candidate feature to predict Y, SET approximate the best augmentation candidate by finding the one that best predicts the residual of Y. After identifying the optimal augmentation candidate, f_{opt}, for each iteration, SET first augment T with f_{opt} and train a linear regression model to predict Y. Then it updates Y to the residual of the linear regression model, and finally re-calibrate the semi-ring structure of T (L12 in Algorithm 2).

This approach simplifies multi-variable linear regression models to incremental simple linear regression models throughout the search. The training and evaluation of each augmentation candidate is further reduced to vector inner products followed by simple algebraic operations. For concreteness, consider the right half of Figure 3 for an illustrating example. With this optimization, the number of unique $\sum f_i f_j$ to be computed for each augmentation candidate is constrained to a fixed constant. Moreover, it completely avoids the expensive matrix inverse operation $(\mathbf{X}^T\mathbf{X})^{-1}$. We empirically show that the quality of the final augmentation plan $\mathcal{P}(T)$ with approximation using linear regression to predict Y has similar accuracy (measured in r^2) as compared to that without approximation, while reducing the runtime by over 100x. We leave a rigorous proof of the accuracy guarantee to future work.

Algorithm 3 GPU Vertical Augmentation Search Algorithm

1: **Input:** $cluster_1, cluster_x, cluster_{x^2}, user_1, user_y, user_{y^2}$
2: $cov_{x^2} \leftarrow \text{sum}(cluster_{x^2} \times user_1, \text{dim} = 0)$ ▷ $\times$ represents matrix multiplication
3: $cov_x \leftarrow \text{sum}(cluster_x \times user_1, \text{dim} = 0)$
4: $cov_1 \leftarrow \text{sum}(cluster_1 \times user_1, \text{dim} = 0)$
5: $cov_{xy} \leftarrow \text{sum}(cluster_x \times user_y, \text{dim} = 0)$
6: $cov_{y^2} \leftarrow \text{sum}(cluster_1 \times user_{y^2}, \text{dim} = 0)$
7: $cov_y \leftarrow \text{sum}(cluster_1 \times user_y, \text{dim} = 0)$
8: $cov_{\bar{x}} \leftarrow cov_x / cov_1$ ▷ element-wise division between vectors
9: $cov_{\bar{y}} \leftarrow cov_y / cov_1$
10: $S_{xx} \leftarrow cov_x^2 - 2cov_{\bar{x}} \times cov_x + cov_1 \times cov_{\bar{x}}{}^2$ ▷ sum of square deviation of x
11: $S_{xy} \leftarrow cov_{xy} - cov_{\bar{x}} \times cov_y - cov_x \times cov_{\bar{y}} + cov_1 \times cov_{\bar{x}} \times cov_{\bar{y}}$ ▷ sum of cross-deviations of x and y
12: $slope \leftarrow S_{xy} / S_{xx}$
13: $intercept \leftarrow cov_{\bar{y}} - slope \times cov_{\bar{x}}$
14: $TSS \leftarrow cov_{y^2} - 2cov_{\bar{y}} \times cov_y + cov_1 \times cov_{\bar{y}}{}^2$
15: $RSS \leftarrow cov_y^2 + cov_1 \times intercept^2 + slope^2 \times cov_x^2 - 2(slope \times cov_{xy} + intercept \times cov_y - slope \times intercept \times cov_x)$
16: $r^2 \leftarrow \vec{1} - (RSS/TSS)$
17: **Return:** $\text{argmax}(r^2)$ ▷ return the index of f_{opt}

To this end, we may expand line 5-9 of Algorithm 2 (highlighted in blue) with all our optimization techniques, as shown in Algorithm 3. When processing input data, each iteration of the greedy search begins with identifying the cluster with overlapping join key domains. SET then preprocess the residual into three tensors indexed by the same join key domain, each has dimension $m \times 1$. For each iteration of search for best feature (L5-9 in Algorithm 2), we employ GPU acceleration (Algorithm 3) to parallelize the computation shown in Figure 3.

L2-7 computes various semi-ring aggregates over join. L8-13 calculates statistics for a simple linear regression model. This involves computing means for the explanatory and target variables (denoted as x and y), and then determining the sum of squares (L10) and cross-deviations (L11), which are essential for deriving the slope and intercept of a simple linear regression model. L14-16 computes the (adjusted) r^2 value. L17 finds the feature with the highest r^2 value, indicating the feature with the most explanatory power in the regression model. As shown in Figure 2, our results demonstrate that this approach allows us to evaluate each iteration with $\sim$ 1.4 seconds for > 1M features over a join key domain of $\sim$ 500.

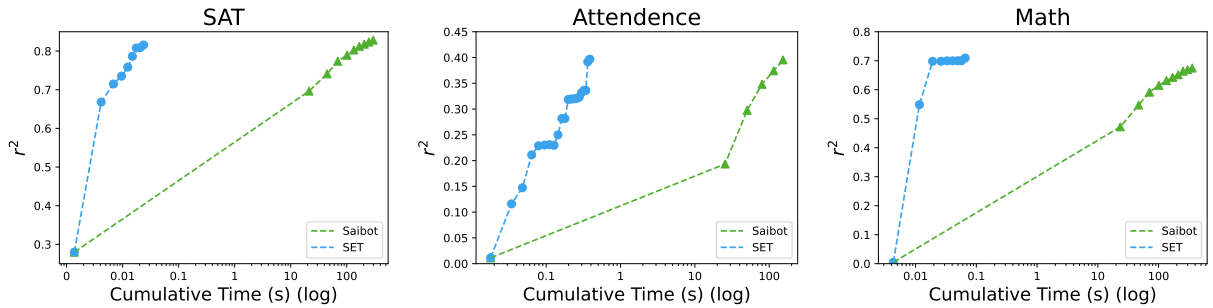

Figure 4: Accuracy of a linear regression model evaluated on $\mathcal{P}(T)$, SET finds an augmentation plan with similar accuracy as compared to that discovered by Saibot while Saibot require 100x more time.

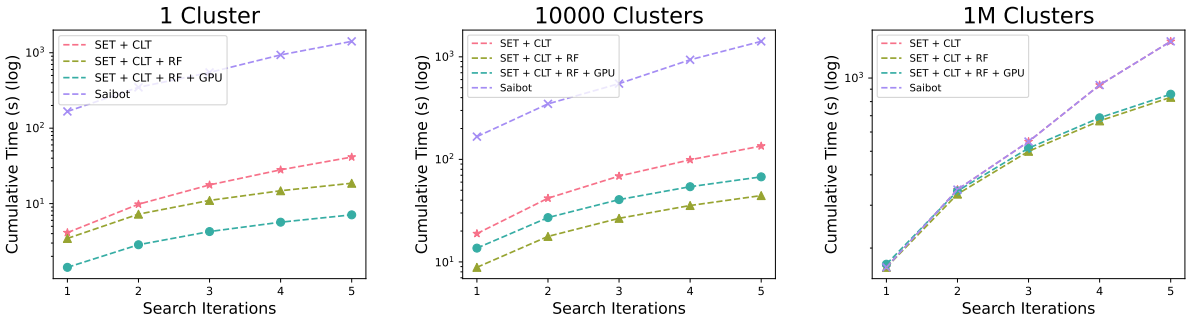

Figure 5: Cumulative runtime with respect to search iterations for different combinations of optimization techniques.

4 EXPERIMENT

Our evaluation aims to understand two main questions: **Q1**: How effectively does the residual-fitting approximation achieve its intended accuracy? and **Q2**: To what extent do the optimization techniques introduced in Section 3 reduce search latency?

4.1 Q1: Accuracy

Data and Workload. We aim to evaluate the accuracy guarantees of our proposed method. We collected 322 datasets from NYC open data [4] and 741 datasets from Data.gov [8]. Among all datasets, we form 4 different clusters indexed by popular join key domains:

- **Year** has a domain size of 265 with 2656 joinable features.
- **Borough** has a domain size of 58 with 1357 joinable features.
- **District** has a domain size of 923 with 1105 joinable features.
- **DBN** has a domain size of 6423 with 5168 joinable features.

We then create three supervised learning tasks using the following 3 datasets:

- **SAT** [3] contains 2012 SAT scores, and can be joined with features indexed by **DBN**.
- **Attendence** [2] contains 2013-19 school attendance data and can be joined with features indexed by **Year** and **DBN**.
- **Math** [1] contains 2013-18 Math grades and can be joined with features indexed by **Borough** and **DBN**.

To create supervised learning tasks, we adopt a leave-one-out strategy: choose 1 of the 1063 datasets from NYC open data and Data.gov as the input and use the remaining datasets to form the relational data repository for SET to search from.

Figure 4 shows cumulative latency with respect to the r^2 score of the augmentation plan, evaluated by training a linear regression model using all features in $\mathcal{P}$. Empirical results reveal that the r^2 score obtained from a linear regression model, when trained on the augmentation set identified by SET, has equivalent performance as that discovered by Saibot.

4.2 Optimization Effectiveness

One key tuning knob for our optimization is the number of clusters in the relational data repository. Our goal is to understand how different number of clusters impact the performance of SET's algorithm. We create synthetic datasets containing 1M numerical features, and vary the number of clusters ($n_{clt} = \{1, 10000, 1M\}$) by generating n_{clt} synthetic join keys for each cluster, respectively. Then, we construct an input dataset that includes a target variable along with n_{clt} different join keys, corresponding to each cluster. We report the latency over 5 iterations of the augmentation search with respect to different combinations of optimization techniques.

Figure 5 shows the cumulative latency with respect to 5 search iterations. For one single cluster containing 1M features, by clustering based on join key domains (CLT), SET the cumulative runtime for more than 10x as compared to Saibot [12]. By incorporating residual fitting (RF), we observe a stabilization in latency across iterations. Additionally, the use of GPUs further cuts the cumulative runtime by another 10x. However, with 10,000 clusters, the benefits of our optimizations diminish, while SET is still >50x faster than Saibot [12]. In the extreme case where each cluster only contains 1 feature, residual fitting *RF* guarantees constant latency across search iterations, whereas the advantage of our hardware optimization vanishes.

5 CONCLUSION

SET is a data augmentation search platform that searches large relational data repositories to find join-compatible datasets to improve supervised learning performance. SET employs both hardware and algorithmic optimizations by introducing a novel approximation algorithm that transform the search process into matrix operations, which fits perfectly with the architecture of GPUs. SET is able to achieve 1.4s latency in finding the optimal feature from over 1M features, with >100x faster than existing techniques.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1845638, 1740305, 2008295, 2106197, 2103794, and support from Amazon, Google, Adobe, and CAIT. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funders.

Acknowledgements: This material is based upon work supported by the National Science Foundation under Grant No. 1845638, 1740305, 2008295, 2106197, 2103794, 2312991, Amazon, Google, Adobe, and CAIT.

REFERENCES

[1] [n.d.]. 2013 -2018 School Math Results. https://data.cityofnewyork.us/Education/2013-2018-School-Math-Results/m27t-ht3h.
[2] [n.d.]. 2013-2019 Attendance Results - School. https://data.cityofnewyork.us/Education/2013-2019-Attendance-Results-School/vww9-qguh/.
[3] [n.d.]. New York City school level College Board SAT results for the graduating seniors of 2010. https://data.cityofnewyork.us/Education/SAT-scores/vtmi-3hwp.
[4] 2022. NYC Open Data. https://opendata.cityofnewyork.us/.
[5] 2023. Google Dataset Search. https://datasetsearch.research.google.com/.
[6] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2016. FAQ: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 13–28.
[7] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: automatic relational data augmentation for machine learning. *arXiv preprint arXiv:2003.09758* (2020).
[8] Data.gov. 2024. Data.gov: The Home of the U.S. Government's Open Data. https://www.data.gov/
[9] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.
[10] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. METAM: Goal-Oriented Data Discovery. *arXiv preprint arXiv:2304.09068* (2023).
[11] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.
[12] Zezhou Huang, Jiaxiang Liu, Daniel Alabi, Raul Castro Fernandez, and Eugene Wu. 2023. Saibot: A Differentially Private Data Search Platform. *arXiv preprint arXiv:2307.00432* (2023).
[13] Zezhou Huang, Jiaxiang Liu, Haonan Wang, and Eugene Wu. 2023. The Fast and the Private: Task-based Dataset Search. *arXiv preprint arXiv:2308.05637* (2023).
[14] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data acquisition for improving machine learning models. *Proceedings of the VLDB Endowment* 14, 10 (2021), 1832–1844.
[15] Natalia Miloslavskaya and Alexander Tolstoy. 2016. Big Data, Fast Data and Data Lake Concepts. In *Procedia Computer Science*. Elsevier.
[16] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A sketch-based index for correlated dataset search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2928–2941.
[17] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: a fast and lightweight AutoML Library. *Proceedings of Machine Learning and Systems* 3 (2021), 434–447.