Benchmark of DNN Model Search at Deployment Time

Lixi Zhou
Arizona State University
Tempe, USA
lixi.zhou@asu.edu

Arindam Jain
Arizona State University
Tempe, USA
ajain243@asu.edu

Zijie Wang
Arizona State University
Tempe, USA
zijiewang@asu.edu

Amitabh Das
Arizona State University
Tempe, USA
adas59@asu.edu

Yingzhen Yang
Arizona State University
Tempe, USA
yyang409@asu.edu

Jia Zou
Arizona State University
Tempe, USA
jia.zou@asu.edu

ABSTRACT

Deep learning has become the most popular direction in machine learning and artificial intelligence. However, the preparation of training data, as well as model training, are often time-consuming and become the bottleneck of the end-to-end machine learning lifecycle. Reusing models for inferring a dataset can avoid the costs of retraining. However, when there are multiple candidate models, it is challenging to discover the right model for reuse. Although there exist a number of model sharing platforms such as ModelDB, TensorFlow Hub, PyTorch Hub, and DLHub, most of these systems require model uploaders to manually specify the details of each model and model downloaders to screen keyword search results for selecting a model. We are lacking a highly productive model search tool that selects models for deployment without the need for any manual inspection and/or labeled data from the target domain. This paper proposes multiple model search strategies including various similarity-based approaches and non-similarity-based approaches. We design, implement and evaluate these approaches on multiple model inference scenarios, including activity recognition, image recognition, text classification, natural language processing, and entity matching. The experimental evaluation showed that our proposed asymmetric similarity-based measurement, adaptivity, outperformed symmetric similarity-based measurements and non-similarity-based measurements in most of the workloads.

CCS CONCEPTS

• **Information systems** → **Clustering and classification**; *Evaluation of retrieval results*; • **Computing methodologies** → **Neural networks**.

ACM Reference Format:
Lixi Zhou, Arindam Jain, Zijie Wang, Amitabh Das, Yingzhen Yang, and Jia Zou. 2022. Benchmark of DNN Model Search at Deployment Time. In *34th International Conference on Scientific and Statistical Database Management (SSDBM 2022), July 6–8, 2022, Copenhagen, Denmark*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3538712.3538725

SSDBM 2022, July 6–8, 2022, Copenhagen, Denmark

ACM ISBN 978-1-4503-9667-7/22/07…$15.00
https://doi.org/10.1145/3538712.3538725

1 INTRODUCTION

Reusing a pre-trained model rather than finetuning or retraining can avoid the substantial efforts required for collecting and annotating training data, and will greatly simplify the model deployment costs [45]. However, when there are multiple candidate models, the prediction of the serving accuracy of a model in a target query dataset is inherently a complex problem. It is related to a number of factors, such as the model architecture, initial weights, hyperparameters, the similarity and adaptivity between the source training dataset and target query datasets [19], etc. In this work, we focus on a broad class of model versioning scenarios, where the users need to choose a few models from a large repository of models that have similar architectures [35].

Motivating Scenario 1. Marketplace-level Forecasting at Uber. They forecast supply, demand, and other quantities in real time for hundreds of cities across the globe. Uber is operating in many cities across the world, and different cities may pose different geospatial characteristics [40]. Besides, the Uber business might be at different growth stages for different cities. Therefore, they divide the problem spatially by city, training a model instance for each city-quantity combination using the same model architecture. In addition, they frequently update the models through retraining, when they detect model performance degradation due to the changing market conditions, and they need to independently trigger the retraining of the models for a city. All of these lead to thousands of or even more model versions [40].

Motivating Scenario 2. Downtime Prediction at Manufacturers. A steel plate manufacturer [44] owns several plants in different countries. The data of machines from different locations are collected centrally. This includes time-series data, such as temperature, humidity, and vibration data, as well as high-resolution image data from cameras. Bob wants to create a machine learning model that predicts the downtimes for a specific machine type. However, the failure situation is changing from time to time, therefore Bob retrains and redeploys the model from time to time to keep it up-to-date, thus leading to many different versions.

Model versioning has motivated a number of model management platforms, such as Gallery [40], ModelDB [42], ModelHub [31], Metadata Tracking [38], MLFlow [12], TensorFlow Hub [4], PyTorch Hub [3], and DLHub [1]. However, the discovery of the proper model versions for serving is mostly through the intuition of domain experts and trial-error processes. Most of these platforms provide a model search tool based on the model metadata as shown in Tab. 1.

Table 1: Model search based on model metadata

Search Category	Filter Examples
Publication Schemas	author, date, description
Model Information	algorithm, software version, network architecture, dependency
Development Provenance	versions, contributors
Training Information	training datasets, parameters
Performance	accuracy, recall, precision, evaluation dataset, evaluation date

However, none of the above information can directly tell *which version(s) of a model should be selected for inference on a new dataset*, which is exactly the problem that we will address in this study. We observed that existing model serving platforms lack an efficient mechanism to search for one or a few model versions that will have the best inference accuracy on the target query dataset. Instead, users need to perform a text-based search using keywords and scrutinize each search result by deploying and testing the model. Such human-centered model selection based on trial and error is inefficient, which delays the model deployment and incurs significant human costs. To alleviate such overheads and human costs, it is urgent to automate this process.

Existing neural architecture search or AutoML [18, 23, 34, 41] cannot solve the problem as well. That's because these techniques are designed for the training stage rather than the deployment stage. At the deployment stage, we do not have the labels of the testing data, so it is not practical to retrain a model from scratch. Also, existing AutoML techniques do not consider how to reuse pre-trained models, which are only available after the training stage.

In this work, we proposed and compared multiple model search strategies that are designed for the deployment stage, as follows:

1. Symmetric data similarity-based approach. A basic assumption in learning theory is that the training and testing sets are drawn from the same probability distribution [19]. If the probability distribution of the testing data is different from the training dataset, the pattern induced from the training dataset may not be relevant to the testing data, and thus such difference will result in poor prediction accuracy [28].

Therefore, if the pre-trained model's source domain (i.e., the domain of the training data) has similar probability distribution with the target domain (i.e., the domain of the testing data), the pre-trained model may achieve good accuracy in the target domain. This motivates a nearest neighbor search approach that selects the models, of which the training datasets are most similar to the target query dataset.

However, given a large number of models and large size of training data (as illustrated in Fig. 1), pair-wise comparison of all models' training datasets to the target query dataset is not practical for high-frequency model search requests [40]. For example, existing studies [2, 5] show that if search response times increase from 1 to 4 seconds, user experience, as well as conversion rates, drop sharply.

To reduce the pair-wise comparison overhead, we consider leveraging Locality Sensitive Hash (LSH), which is a popular technique to solve nearest neighbor problems. Recently, the LSH for measuring the similarity of probability distributions, such as JS-divergence is studied [13, 30], which is applied in this work and called JSD-LSH.

2. Asymmetric data similarity-based approaches. The inference accuracy is asymmetric. For example, a dataset **A** may contain only a subset of instances of another dataset **B**; then supposing we have a model $M_{\mathbf{A}}$ that is trained on **A**, and a model $M_{\mathbf{B}}$ that is trained on **B**, the accuracy of $M_{\mathbf{A}}$ on **B**, and the accuracy of $M_{\mathbf{B}}$ on **A** could be very different. Because of such asymmetry, it is problematic to simply apply the symmetric similarity measurement to the features of the training dataset and the features of the target query dataset.

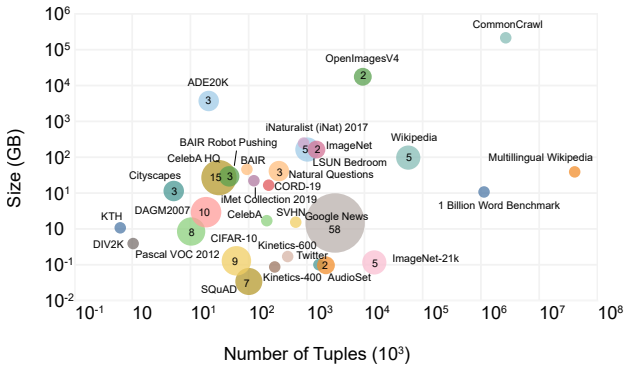


Figure 1: TensorFlow Hub Models. The value and size of the circle denote the number of models trained on the dataset.

To address the problem regarding the asymmetric serving accuracy on the swapping of source and target domains, we propose a novel similarity measurement called `adaptivity`. It first requires partitioning the training dataset and the target query dataset into multiple partitions of the same size respectively. Then, the `adaptivity` is computed as the set containment of the training dataset to the target query dataset, which is defined as the ratio of the number of similar partitions to the number of the partitions in the target query dataset. A great benefit of the `adaptivity` metric is that it can be converted to Jaccard similarity [8], which motivates us to use the Minwise LSH to accelerate the pair-wise set containment computations. However, the problem is that Minwise LSH is designed for indexing high-dimensional vectors, but in our case, each dataset is a vector of partitions, and it is hard to directly derive the Minwise LSH for such a complicated vector, lacking an efficient comparator for two partitions. To address the problem, we further propose a novel two-level LSH index. The top level is the aforementioned Minwise LSH index, and the bottom level is the aforementioned JSD-LSH. Utilizing the JSD-LSH, each partition is converted to a JSD-LSH signature, and two partitions have the same signature if they have similar probability distributions (i.e., small JS-divergence). By seeing each signature band as a value, a dataset is abstracted as a vector of values, from which Minwise LSH can be easily computed for estimating the `adaptivity`.

3. Non-similarity-based Approaches. When the schemas of the training datasets of candidate models are inconsistent with target datasets, we need to compute the overlapping of the source and target datasets at runtime, because such overlap cannot be predicted and precomputed before an ad-hoc model search task is issued. In such a situation, we need to store the training dataset of each candidate model to facilitate such runtime overlap computing, which brings significant storage overheads and complexity in managing such datasets. Therefore, in this study, we also considered two non-similarity-based approaches to avoid such issues. The first approach leverages the predictions of all candidate models to vote for the label of a targeting sample and select the model whose predictions

are mostly voted as the best model. While this approach shares some similarities with ensemble inferences, its purpose is to select the best model for deployment, rather than making inferences. The second approach simply selects the model that has the best accuracy in its source domain from which it is trained, regardless of the target domain.

Our Key Contributions are summarized as follows:

(1) In this work, we identified the problem of searching models for serving a target dataset from a set of candidate models that are trained using a similar methodology with different training data. Our work will greatly enhance the capability of existing model repositories and improve the productivity of the model reuse process, which is urgently needed in production.

(2) We proposed various model search methodologies that do not rely on any labeled datasets, including the approaches based on various symmetric measurements about the similarity between the source domain and target domain, the approaches based on asymmetric similarity measurements, and the non-similarity-based approaches such as the voting approach, and the approach solely based on source accuracy.

(3) We implemented the proposed approaches and conducted extensive experiments to compare their effectiveness and efficiency using five different applications: activity recognition, image recognition, text classification, entity matching, and natural language processing (NLP). Our study has shown that our proposed asymmetric similarity-based measurement, adaptivity, outperformed symmetric similarity-based measurements and non-similarity-based measurements in most of the workloads.

1.1 Key Observations

We summarized our key observations of various model selection strategies at deployment time, not using any labeled data, as follows:

(1) Asymmetric similarity-based measurements such as adaptivity outperformed symmetric similarity-based measurements such as JS-divergence and L2 distance for most of the workloads except the text classification workload.

(2) Similarity-based measurements, such as adaptivity, JS-divergence, and L2 distance, achieved a significantly lower error rate than non-similarity-based measurements in most of the workloads except the text classification workload.

(3) The similarity-based measurements require the precomputation of LSH signatures. The source-accuracy-based measurement requires the collection of the training accuracy of candidate models. The voting approach does not require any preprocessing, but it incurs the most runtime computational overheads for running each candidate model on the target dataset.

(4) When the schemas of candidate models are inconsistent with the schema of the target dataset (e.g., using a different dictionary), we need to store the training datasets of candidate models for similarity-based measurements to compute the overlapping features between source training datasets and the target inference datasets at runtime, which incurs significant storage overhead and management complexity. The non-similarity-based measurements will not have such a problem.

2 BACKGROUND

2.1 Jensen-Shannon (JS) Divergence

JS-divergence [27] is a measurement of the similarity between two probability distributions, which is a symmetric metric derived from the asymmetric Kullback-Leibler divergence [22].

Let P and Q be two probability distributions associated with a common sample space Ω, and let $M = (P+Q)/2$. The JS-divergence is defined by:

$$D_{JS}(P \parallel Q) = \frac{1}{2}D_{KL}(P \parallel M) + \frac{1}{2}D_{KL}(Q \parallel M) \tag{1}$$

Here, D_{KL} denotes the Kullback-Leibler divergence, which is defined as the following for discrete distributions:

$$D_{KL}(P \parallel Q) = \sum_{\mathbf{x}\in\Omega} P(\mathbf{x}) \log(\frac{P(\mathbf{x})}{Q(\mathbf{x})}), \tag{2}$$

and as following for continuous distributions:

$$D_{KL}(P \parallel Q) = \int_{\Omega} P(\mathbf{x}) \log(\frac{P(\mathbf{x})}{Q(\mathbf{x})})\, \mathrm{d}\mathbf{x} \tag{3}$$

We focus on using JS-divergence to calculate the similarity of two datasets in this paper. The reasons are: First, JS-divergence is a widely used similarity metric for probability distributions [24]. Second, it is easier to find LSH schemes for JS-divergence, compared to other similarity measurements of probability distributions [30].

While other metrics such as Maximum Mean Discrepancy (MMD) could measure the domain adaptivity more accurately, it requires an optimization process and significantly higher computational costs, as illustrated in Tab. 2.

Table 2: Average latency comparison on Activity Recognition (Sec. 7.1) datasets (Unit: seconds).

Jaccard similarity	KL-divergence	JS-divergence (w/o LSH)	MMD
379	29	87	6451

2.2 Locality Sensitive Hashing (LSH)

LSH was developed for the general approximate nearest neighbor search problem [20]. It requires a family of LSH functions, each of which is a hash function whose collision probability increases with the similarity of the inputs. The (r_1, r_2, p_1, p_2)-sensitive LSH family is formally defined in Definition. 2.1.

Definition 2.1. Let $\mathcal{F} = \{h : M \rightarrow U\}$ be a family of hash functions for distance measurement D. $\mathcal{F}$ is (r_1, r_2, p_1, p_2)-sensitive ($r_1 < r_2$ and $p_1 > p_2$), if $\forall p, q \in M$, it satisfies that: (1) if $D(p,q) \leq r_1$, we have $Pr[h(p) = h(q)] \geq p_1$; (2) if $D(p,q) \geq r_2$, we have $Pr[h(p) = h(q)] \leq p_2$.

LSH is first proposed by Indyk et al. [20] for measuring the Hamming distance in a d-dimensional Euclidean space, which requires the data to be vectors with fixed dimensions. MinHash [8] is a widely used family of hash functions for Jaccard similarity. SimHash is an LSH scheme for Cosine distance [10]. Both Minwise LSH and SimHash are only applicable to high-dimensional binary vectors or sets of values (without fixed dimensions), and MinHash is usually considered to be more computationally efficient than SimHash [52].

2.2.1 Minwise LSH. Minwise LSH [8] was proposed as an index for estimating the Jaccard similarity between two sets of values. It converts each set into a MinHash signature using a family of Minwise hash functions. Each Minwise function hashes each value in the set into an integer and returns the minimum hash value.

Given a Minwise hash function h_{min}, it has been proved that the probability of the two minimum hash values, which are observed over two sets of values X and Y, equal to each other, is the Jaccard similarity of X and Y: $P[h_{min}(X) = h_{min}(Y)] = jaccard(X, Y) = \frac{X \cap Y}{X \cup Y}$ [8]. Furthermore, usually, a family of K Minwise hash functions are applied to a set of values, generating K minimum hash values as a signature. Given the signatures of X and Y, Jaccard similarity can be estimated as the ratio of the number of collisions in the corresponding minimum hash values to K. In practice, for convenience, the K values are further partitioned into L bands, so that the Jaccard similarity can be estimated as the ratio of the number of bands from the two sets that are equivalent to the total number of bands. The lookup of matching bands can be accelerated by storing the Minwise signatures of all candidates corresponding to each band into a hash table.

2.2.2 LSH for JS-divergence. As mentioned, the LSH schemes for probability distributions are recently studied [13, 30]. S2JSD-LSH [30] provides LSH functions for a measurement that approximates the square root of two times the JS-divergence. Unfortunately, their LSH design doesn't provide any bound on the actual JS-divergence. Then Chen et al. [13] propose a new LSH scheme for the generalized JS-divergence through the approximation of the squared Hellinger distance, which is proved to be bounded with the actual JS-divergence and thus used in this work, as defined in Eq. 4:

$$h_{\mathbf{a},b} = \lceil \frac{\mathbf{a} \cdot \sqrt{P} + b}{r} \rceil \tag{4}$$

where P is a probability distribution in the sample space Ω, $\mathbf{a} \sim \mathcal{N}(0, I)$ is a $|\Omega|$-dimensional standard normal random vector, $\cdot$ represents the inner product operation, b is a random variable uniformly distributed on $[0, r]$, and r is a positive real number. This approximation is proved to be lower bounded by a factor 0.69 for the JS-divergence [13].

3 PROBLEM DEFINITION

We are given a list of models of similar architecture, each of which is trained using a different training dataset that shares the same schema of features and labels (e.g., generated from similar data pipelines in the same organization). The set of training datasets for these models is denoted as $T = \{t_1, ..., t_k\}$. A training dataset for the i-th model is denoted as $t_i = \{< \vec{x}, y >\}$, where each training sample consists of a feature vector $\vec{x}$, and a label y. Similarly, a target dataset is denoted as $\mathbf{I} = \{< \vec{x'}, y' >\}$, where each testing sample consists of a feature vector $\vec{x'}$, which is known, and a label y', which is unknown and needs to be inferred. The schema of the target dataset is consistent with the training datasets. The question is which model should be selected so that it will achieve the best inference accuracy on $\mathbf{I}$.

While in this work, we focus on the above-formalized problem, our proposed approaches can be extended to handle source and the target datasets that have similar but inconsistent schemas, by computing their overlapping feature space and applying the proposed approaches to the overlapping features.

4 SYMMETRIC SIMILARITY MEASUREMENTS

In this work, we mainly consider two representative and widely used symmetric similarity measurements: (1) JS-divergence, which models the source and target datasets as two probability distributions and measures their JS-divergence. (2) Euclidean distance (L2 distance), which computes the distance between the centers of the source and target datasets.

In a model search scenario based on JS-divergence, we need first obtain the target dataset, and the (source) training dataset used by each model. Then each source training dataset will be compared to the target dataset. This process not only incurs significant computational overheads but also brings additional complexity, such as storage overheads and privacy issues, for managing the training dataset of each model.

To address these issues, we can leverage the LSH for JS-divergence, which is recently investigated by Chen et al. [13]. We precompute the JS divergence-LSH (JSD-LSH) signature, as described in Sec. 2.2.2, for each dataset. Only the JSD-LSH signatures, rather than the training datasets, will be stored in our proposed system, which will not only alleviate the computational overheads but also reduce the storage overheads and avoid privacy issues.

Similarly, when leveraging the Euclidean distance, the center of each training dataset needs to be precomputed and stored. If there exists a large number of models trained on high-dimensional datasets, LSH for Euclidean distance can also be leveraged to accelerate the searching process.

As discussed in Sec. 2, we do not consider Jaccard similarity or Minwise LSH, mainly because they are designed for binary vectors or categorical datasets. However, feature vectors used by deep learning are mostly numerical. We do not consider Maximum Mean Discrepancy (MMD), which is a well-known symmetric similarity measurement for two probabilistic distributions, mainly because it incurs significantly higher computational overhead as illustrated in Tab. 2.

5 ASYMMETRIC SIMILARITY MEASUREMENTS

In this section, we first explain why symmetric metrics that measure the similarity of two sets of features cannot be used to estimate whether a model trained in one set will effectively infer the other dataset. Then, to address the issue, we propose a new asymmetric `adaptivity`. We further propose a novel two-level LSH index framework to accelerate the system-wide computation of `adaptivity` with theoretical proofs.

Why symmetric similarity metrics will not work for this problem? A straightforward idea is to simply measure the similarity between the set of features to be inferred, denoted as $F_I = \{(\vec{x'})\}$, and the set of features of each training dataset, denoted as $F_i = \{(\vec{x})\}$, and select the model that has the most similar training dataset. However, if the similarity measurement is symmetric, there

exists a significant problem that the results will be skewed by the discrepant sizes of the training dataset and target dataset. For example, if $F_I \subset F_i$, the model trained on t_i may have good accuracy on **I**, because the model has seen all of the samples in **I** during the training process. On the contrary, the model trained on **I** may have relatively worse accuracy on F_i, because the model has only seen a portion of samples in F_i in training. Therefore, it's more important to measure the ratio of the similar samples to the samples in the target dataset, which is asymmetric.

5.1 Adaptivity Measurement

While an asymmetric similarity measurement such as KL-divergence may also address the issue, there is no existing work that has discussed the LSH for measuring KL-divergence. Therefore, the overhead incurred by the pairwise computation of KL-divergence is prohibitive. In this work, we propose a new measurement, called `adaptivity`, which can be converted to Jaccard similarity, a symmetric similarity measurement, and thus its computation can be accelerated using a novel two-level LSH (See Sec. 5.2).

The idea is to first randomly partition F_i and **I** into partitions of the same size and then compute a set containment measurement that is defined as the ratio of the number of similar partitions shared by a training dataset and the target inference dataset to the total number of partitions in the target dataset. The formal definition of the metric is as follows:

Definition 5.1. Supposing we have two datasets $\mathcal{S}_s = \{p_0, ..., p_m\}$ and $\mathcal{S}_t = \{q_0, ..., q_n\}$, where $p_i (0 \leq i < m)$ and $q_j (0 \leq j < n)$ are partitions that have equivalent sizes (except for the last residue partition), given a threshold t, we can join $\mathcal{S}_s$ and $\mathcal{S}_t$ to identify a subset of $\mathcal{S}_t$, denoted as $\mathcal{S}_t^*$, where $\forall q_j \in \mathcal{S}_t^*, \exists p_i \in \mathcal{S}_s$, satisfying that $D_{JS}(p_i, q_j) \leq t$. We denote the total number of partitions in $\mathcal{S}_t^*$ as $|\mathcal{S}_t^*| = l$. Then we can derive a new metric to measure the adaptivity as the set containment of the target dataset ($\mathcal{S}_t$) in the source dataset ($\mathcal{S}_s$), denoted as: $adaptivity(\mathcal{S}_s, \mathcal{S}_t) = \frac{\mathcal{S}_s \cap \mathcal{S}_t}{\mathcal{S}_t} \sim \frac{\mathcal{S}_t^*}{\mathcal{S}_t} = \frac{l}{m}$.

Based on the definition of the `adaptivity` metric, we further formulate the problem as illustrated in Def. 5.2, which can be easily extended to a nearest neighbor search problem of finding the top-k models that have the highest adaptivity to the target domain.

Definition 5.2. We have a database of n pre-trained models (i.e., source tasks in a source domain), represented as $\mathcal{M} = \{M_1, ..., M_n\}$. Each model $M_i \in \mathcal{M}$ has a source domain $\mathcal{S}_i$, which is a set of training instances that have the same types of features. We have a target query dataset, q, that is associated with a target domain $\mathcal{T}_q$, which has the same types of features as source domains. Given an `adaptivity` threshold t', the database should return a set of relevant models, denoted as $\mathcal{M}_q \subset \mathcal{M}$, so that $\forall M_i \in \mathcal{M}_q, adaptivity(\mathcal{S}_i, \mathcal{T}_q) \geq t'$.

Justification of the Adaptivity Metric. It is a common practice of leveraging partitioning to improve the accuracy of LSH in an asymmetric scenario [52]. Based on partitioning, the set containment measurement is asymmetric, indicating the ratio of the number of matched partitions to the total number of partitions in the *target* dataset. Thus, it can effectively represent the asymmetric relationship between the source domain and the target domain. In addition, this set containment measurement and Jaccard similarity can be transformed to each other [52], which facilitates the determination of the threshold t'. In addition, because Minwise hash is based on Jaccard similarity (Sec. 2.2.1), we can combine the Minwise hash and the JSD-LSH (for measuring) to facilitate the computation of the adaptivity metric, as described in detail in the next section.

5.2 Novel Two-Level LSH Index

In this section, in order to accelerate the computation of the adaptivity, we propose a novel two-level LSH index that uses JSD-LSH [13] for measuring the similarity between partitions and uses Minwise LSH [8] for measuring the adaptivity from the source dataset to the target dataset.

5.2.1 JSD-LSH. We first consider the sensitiveness of JSD-LSH applied to each partition. The hash functions $\{h_{\mathbf{a},b}\}$, defined in Eq. 4, form a $(t, c^2\frac{U(\lambda)}{L(\lambda)}t, e_1, e_2)$-sensitive family for the generalized JS-divergence with parameter λ (it reduces to usual JS-divergence if $\lambda = 0.5$) [13]. This means, we can leverage $Pr[h(p_i) = h(q_j)] \geq e_1$, to determine whether $D_{JS}(p_i, q_j) \leq t$. We simply denote this mapping relationship as $e_1 = g(t)$. Please see [13] for the proof and more details. In practice, we use a family of K JSD-LSH functions by randomly choosing the hyperparameters $\mathbf{a}$ and b. Then we will obtain K values for each partition by applying the K functions. These K values are further partitioned into L bands. We estimate the probability that p_i and q_j are similar (i.e., $Pr[h(p_i) = h(q_j)]$) as the ratio of the matched JSD-LSH bands to L. If we see each JSD-LSH signature as a set of JSD-LSH bands, the $Pr[h(p_i) = h(q_j)]$ is actually the Jaccard similarity of the two sets. This means $jaccard(h(p_i), h(q_j)) \geq g(t)$, which connects the value of the JS-divergence threshold t as defined in Def. 5.1 and the Jaccard similarity threshold t' defined in Def. 5.2, as we explained in the next section.

5.2.2 Minwise LSH. Now we consider the adaptivity threshold t', given source dataset $\mathcal{S}_s = \{p_0, ..., p_m\}$, and target dataset $\mathcal{S}_t = \{q_0, ..., q_n\}$, Jaccard similarity is defined as $jaccard(\mathcal{S}_s, \mathcal{S}_t) = \frac{S_s \cap S_t}{S_s \cup S_t} = \frac{l}{m+n-l}$, which means $jaccard(\mathcal{S}_s, \mathcal{S}_t) = \frac{adaptivity(\mathcal{S}_s, \mathcal{S}_t)}{\frac{n}{m}+1-adaptivity(\mathcal{S}_s, \mathcal{S}_t)}$.

Therefore, if $adaptivity(\mathcal{S}_s, \mathcal{S}_t) \geq t'$, it means the Jaccard similarity between source domain partitions and target domain partitions satisfies $jaccard(\mathcal{S}_s, \mathcal{S}_t) \geq \frac{t'}{\frac{n}{m}+1-t'}$. We leverage a family of Minwise hash functions as described in Sec. 2.2.1 to generate a Minwise hash signature for each dataset (i.e., regarded as a set of partitions), with each partition represented as a JSD-LSH signature. In addition, as mentioned at the end of Sec. 2.2.2, the equivalence of p_i and q_j is determined by $jaccard(h(p_i), h(q_j)) > g(t)$, where $h(x)$ represents the JSD-LSH function that maps x to a set of L bands, which is denoted as $h(x) = \{b_1, ..., b_L\}$. Therefore, if $g(t) = 1/L$, we can simplify the computation of such recursive Jaccard similarity measurement based on below equation that helps us to derive an upper-bound of the adaptivity metric:

$$jaccard(\mathcal{S}_s, \mathcal{S}_t) \leq L \times jaccard(\cup_{p_i}\{b_i | b_i \in p_i\}, \cup_{q_j}\{b_j | b_j \in q_j\})$$

Proof. The intuition is that if two sets of items (i.e., each item is a JSD-LSH signature computed on a partition) are similar, then if we split each item into sub-items (e.g., each sub-item is a band in a JSD-LSH signature), the two sets of sub-items are still similar to each other, and vice versa. More formally, supposing $\exists A \subset \mathcal{S}_s$ satisfying $\forall p_i \in A, \exists q_j \in \mathcal{S}_t$, and $jaccard(h(p_i), h(q_j)) \geq 1/L$, we have $jaccard(\mathcal{S}_s, \mathcal{S}_t) = \frac{|A|}{|\mathcal{S}_s \cup \mathcal{S}_t|}$. Additionally, $jaccard\ (h(p_i), h(q_j)) \geq 1/L$ indicates that $\exists b_i \in p_i$ and $b_j \in q_j$, satisfying $b_i = b_j$. Therefore, $jaccard(\cup_{p_i}\{b_i|b_i \in p_i\}, \cup_{q_j}\{b_j|b_j \in q_j\}) \geq \frac{|A|}{|\mathcal{S}_s \cup \mathcal{S}_t| \times L}$ Therefore, $jaccard(\mathcal{S}_s, \mathcal{S}_t) \leq L \times jaccard(\cup_{p_i}\{b_i|b_i \in p_i\}, \cup_{q_j}\{b_j|b_j \in q_j\})$, and we can use the latter as an upper-bound estimation of the former metric.

Based on the approximation, a JSD-LSH signature is first generated for each partition and each signature is further split into L bands. Then the set of partitions in a dataset will be flattened into a set of JSD-LSH bands. Given user-specified adaptivity threshold t', we can decide whether two sets have their Jaccard similarity higher than $\frac{t'}{\frac{n}{m}+1-t'}$ times a factor of $L = \frac{1}{g(t)}$ by comparing their Minwise hash signatures as described in Sec. 2.2.1.

One potential problem is that the value of n is probably not a constant shared by all training datasets. To address the problem, we leverage a padding strategy as in asymmetric Minwise LSH [39]. We select an upper bound of n, denoted as n_u, then we pad every training dataset to have n_u partitions by appending $(n_u - n)$ JSD-LSH signatures for non-existing partitions. The values in these padding signatures will not exist in real signatures, so the padding will not affect the precision, while it may lead to false negatives.

The algorithm based on the two-level LSH index to address the problem formulated in Def. 5.2 is illustrated in Alg. 1.

Algorithm 1 Model discovery algorithm

INPUT1: $\mathcal{T}_q$ (query dataset, i.e., the target dataset)
INPUT2: t' (the adaptivity threshold)
INPUT3: L (the number of bands in each JSD-LSH hash)
INPUT4:L_m (the number of bands in each Minwise hash)
INPUT5: $\{hashmap_i\}(1 \leq i \leq L_m)$ (one hashmap for each band of the Minwise-LSH.)
OUTPUT: $\mathcal{M}_q$
$m \leftarrow |\mathcal{T}_q|$
$t^* \leftarrow L \times \frac{t'}{\frac{u}{m}+1-t'}$
$Q \leftarrow partition(\mathcal{T}_q)$
$S \leftarrow \phi$
for $q \in Q$ **do**
  $S \leftarrow S \cup JSDLSH(q)$
end for
$S' \leftarrow MinwiseLSH(S)$ {S' consists of L_m bands}
$\mathcal{M}, \mathcal{M}_q \leftarrow \phi$
for $s_i \in S'$ **do**
  $\mathcal{M}_i \leftarrow hashmap_i.lookup(s_i)$
  $\mathcal{M} \leftarrow M \cup \mathcal{M}_i$
end for
for $M_i \in \mathcal{M}$ **do**
  if $\frac{\mathcal{M}.count(M_i)}{L_m} > t^*$ **then**
    $\mathcal{M}_q \leftarrow \{m\} \cup \mathcal{M}_q$
  end if
end for
return $\mathcal{M}_q$

6 NON-SIMILARITY-BASED APPROACHES

Motivation. When the schemas of the training datasets of candidate models are consistent with the target dataset, the similarity-based approaches can work efficiently, because each dataset can be abstracted as an LSH signature, and only the signature needs to be stored and queried at runtime. However, if we extend the work to a situation where the schemas of the training datasets are not consistent with the target dataset, we need to first identify the overlapping features that are shared by the target dataset and the training dataset of each candidate model. Then we apply the similarity measurements to the overlapped features. Identifying the overlapping features for each candidate model not only requires a lot of computation time but also requires storing all training datasets of candidate models. This may be possible in certain scenarios, e.g., models are natively served from a data management system [21, 46, 50, 53, 54]. However, for general use cases, it is impractical to request users to make their training datasets available for model searching and deployment.

In addition, we found that the similarity measurements only work well when some of the training datasets are correlated with the target datasets. If the training datasets and the target datasets are drawn from two different domains (e.g., the corpus of clinical healthcare records vs. the corpus of movie reviews), similarity-based measurements are not very helpful.

Therefore, to address these limitations of the similarity-based model search approaches, in this study, we also considered two non-similarity-based approaches that do not rely on the training datasets of candidate models, which are described as follows.

1. A Voting Approach. This approach has two phases: *voting* and *selection*. At the beginning of the voting phase, every candidate model has a credit counter. Then, during the voting phase, each candidate model will make a prediction for each sample in the target dataset. For extensions to the situation where the target dataset's schema does not match the candidate model's input feature format, a preprocessing step will be required to convert the target feature to each model's required input format via padding, truncating, reordering, etc.

For each target sample, the prediction that is mostly voted by the candidate models becomes the winner, and all candidate models that make a winning prediction will have their credit counter incremented by one.

In the selection phase, the model that has the maximal number of credits is selected for serving the targeting dataset. While this approach shares some similarities with ensemble inferences, its purpose is to select the best model for deployment, rather than making inferences directly.

2. An Approach based on Source Accuracy. Though the voting approach does not require the storage of the training datasets of the candidate models, the voting process is computation-intensive and if there exists schema inconsistency, the preprocessing of target datasets for voting is difficult to be automated. These limitations motivate a much simpler approach, which selects the model that has the best source accuracy, which is defined as the accuracy

of the model in its source domain from which it is trained (i.e., training accuracy), regardless of the target domain. This approach requires collecting the source accuracy information for each candidate model.

A **qualitative comparison** of all proposed model selection approaches is illustrated in Tab. 3 and Tab. 4.

Table 3: High-Level comparison of model search strategies w/o schema inconsistencies

	Runtime Computational Overheads	Storage Overheads	Preprocessing Overheads
JS-Divergence	JSD-LSH computation	LSH signatures	LSH precomputation
L2 Distance	L2-LSH computation	LSH signatures	LSH precomputation
Adaptivity	Two-level LSH computation	LSH signatures	LSH precomputation
Voting	model prediction	none	none
Source Accuracy	top-k filtering	none	Source accuracy info collection

Table 4: High-Level comparison of model search strategies extended to handle schema inconsistencies

	Runtime Computational Overheads	Storage Overheads	Preprocessing Overheads
Similarity-based approaches	Overlapping & LSH computation	Training datasets	none
Voting	target feature preprocessing & Feature preprocessing	none	none
Source Accuracy	top-k filtering	none	Source accuracy info collection

7 EMPIRICAL EVALUATION

7.1 Workloads and Datasets

We evaluate our proposed methodology using five workloads.

1. Activity Recognition. Human activity recognition (HAR), is to predict the activities (e.g., walking, sitting, running, lying) based on data collected from multiple sensors attached to the human body. HAR is a hot research topic in the pervasive computing area and has been widely applied to indoor localization, sleep state detection, smart home sensing, and virtual reality [6]. In this work, we use three public activity recognition datasets, including OPPORTUNITY [11], PAMAP2 [37], and DSADS [7]. The OPPORTUNITY dataset is collected from four human subjects executing various activities with sensors attached to more than five body parts. The PAMAP2 dataset is collected from nine subjects performing 18 activities with sensors attached to three body parts. The DSADS dataset is collected from eight subjects wearing sensors on five body parts. A dataset is collected for each body part, therefore there are 13 datasets in total. Each dataset has 81 features [43] and 1920 to 5022 samples. Then we apply the aforementioned five model search approaches to a series of scenarios where models are trained on each of these tables and given a target dataset collected from a subject on a specific body part, we want to select a model to achieve the best accuracy on the target dataset.

2. Image Recognition. We randomly sample images from the CIFAR-10 dataset without replacement to create five image datasets with distinct probability distributions. CIFAR-10 is a collection of images with labels of ten classes, with each containing 5,000 images. Then we make four of the subsets skewed by adding 5,000 images of the fourth class to the second and the fourth subset; 5,000 images of the second class to the third subset; 5,000 images of the eighth class to the third subset. These five datasets are called as `Balanced`, `Skewed-1`, `Skewed-2`, `Skewed-3`, `Skewed-4` correspondingly. To evaluate the effectiveness of our proposed model search strategies for the image recognition scenario, we train ResNet56v1 model on each of the subsets respectively and then search for the best model to be reused on each dataset using different strategies.

3. Text Classification (Sentiment). The objective of this task is to classify text as positive or negative sentiment. In this task, we train models on seven different datasets: two 25,000 randomly sampled IMDB Dataset of Movie Reviews; tweets from Twitter which consist of two datasets of 25,00 samples each; Financial PhraseBank contains the sentiments for financial news headlines which has 4,846 samples divided into two datasets; and hate speech tweets dataset which has 4,484 samples. We train models for these tasks using the pre-trained DistilBERT base model (uncased) [1], which contains 6 layers and 40% less parameters than bert-base-uncased. Then for each task, we use one of the seven datasets as the target dataset and try to choose the top- k candidate models to serve on the target dataset using different approaches.

4. Entity Matching (EM). This task is to decide *whether two tuples are referring to the same entity* [9, 16, 17, 26, 49]. We mainly apply DeepMatcher [32], which is an EM tool based on deep learning, to four datasets [2]: Walmart-Amazon, Abt-Buy, DBLP-Scholar, DBLP-ACM; and 11 smaller datasets from the Magellan Data Repository [15] [3]. Each task contains training and testing samples collected from two different datasets. For example, IMDB-TMD is to match movie tuples collected from IMDB and TMD respectively. and IMDB-RottenTomatoes is to match movie tuples from IMDB and Rotten Tomatoes respectively. We focus on the model selection scenarios where a dataset is used for query, and a set of models are trained for the EM tasks on the rest of the datasets as candidates, and then we try to select top-k candidate models to serve the query dataset.

5. Natural Language Processing (NLP). We are mainly interested in two types of NLP tasks. The first task is to identify whether sentence pairs have equivalent semantic meanings. For this task, we train models on three different datasets: Microsoft Research Paraphrase Corpus (MRPC) which includes 3,549 samples; Quora Question Pairs (QQP) which consists of 363,192 samples; and Paraphrase Adversaries from Word Scrambling (PAWS) which has 49,401 samples. The second task is about natural language inference (NLI), which is to identify the textual entailment relationship [29] between sentence pairs. Similar to the first task, we train models on three different datasets: Recognizing Textual Entailment (RTE) which has 2,490 samples; Question NLI, which contains 510,711 samples; and the SCITAIL dataset including 5,302 samples, which is an entailment dataset created from multiple-choice science exams and web sentences. We train models for these tasks using the pre-trained BERT base model [4], which contains 12 layers and 110 millions of

[1] https://tfhub.dev/jeongukjae/distilbert_en_uncased_L-6_H-768_A-12/1
[2] https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md
[3] https://sites.google.com/site/anhaidgroup/useful-stuff/data
[4] https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1

parameters. Then for each task, we use one of the three datasets as the target dataset and try to choose the best model to serve on the target dataset.

7.2 Evaluation Methodology

One major evaluation task is to compare various model searching strategies without labeled data at deployment time. These strategies include:
(1) JS-divergence (a special case of adaptivity when each dataset has only one partition), which is a symmetric similarity measurement for two probability distributions, as mentioned earlier in the paper.
(2) L2-distance, which is a symmetric similarity measurement to get the center of the instances by averaging all feature vectors for the source and target datasets and then computing Euclidean distance between two centers;
(3) Adaptivity, which is an asymmetric similarity measurement to compute the adaptivity of the source dataset to the target dataset;
(4) Voting, which is the number of majority-voted predictions (i.e., the number of predictions approved by a majority of all candidate models) made by a candidate model.
(5) Source accuracy, which is the accuracy of the candidate model on its training dataset. This information is widely available in most types of model databases and in practice, and users usually rely on this information for manual model discovery.

To compare the effectiveness of these approaches quantitatively, we use the following evaluation metrics:
(1) **Pearson correlation coefficient** (PCC) [25], a measure of the linear correlation coefficient between two random variables, to show the correlation of various metrics to the target accuracy (i.e., the prediction accuracy of the candidate model on the target dataset).
(2) **Top-k-error**, is defined as the number of wrong predictions that fail to correctly select the optimal model in the top-k results (ranking of the k selected models is not important here) to the total number of predictions. For example, for the top-3 search, if the ground truth is {C}, the prediction of {B, C, D} is a correct prediction. We use the error rate metric for evaluating the effectiveness of different strategies.

For the `adaptivity` measurement, we tune hyperparameters including the value of r for JSD-LSH, the number of hash functions and the number of bands for Minwise Hash and JSD-LSH, and the number of partitions for each test case by searching the combination of hyperparameters in a pre-specified range.

7.3 Evaluation Results

7.3.1 Activity Recognition. We first evaluate the Activity Recognition workload for the effectiveness of the proposed metrics.
Summary. The comparison of Pearson correlation coefficient values of various metrics, including the adaptivity, the JS-divergence, the L2-distance, and the source accuracy, to the target accuracy are illustrated in Tab. 5. It shows that our proposed adaptivity metric and voting achieve a better correlation with target accuracy compared to other metrics. Also, among the four metrics, the source accuracy exhibits the least relevance to the target accuracy, which verifies our assumption that the similarity between the source and the target plays an important role in determining the target accuracy (i.e., model adaptivity to the target domain). We also compute the top-1, top-2, top-3 error rates, as illustrated in Tab. 6. As mentioned, the Top-k error rate is defined to be the ratio of the number of searches that failed to return all k models correctly to the total number of searches. The results show that `adaptivity` works better than other metrics.

Table 5: Comparison of Pearson correlation coefficient (PCC) for activity recognition (the best Pearson correlation coefficient values are highlighted in bold).

target	pcc of adaptivity	pcc of JS-divergence	pcc of l2-distance	pcc of source-accuracy	pcc of voting
dsads_la	0.61	0.17	0.54	0.44	**0.91**
dsads_ll	**0.72**	0.40	0.68	0.40	0.68
dsads_ra	0.51	0.13	0.54	0.37	**0.78**
dsads_rl	0.58	0.49	0.75	0.58	**0.87**
dsads_t	0.19	0.11	0.42	0.44	**0.80**
oppo_b	0.89	0.66	0.77	0.45	**0.95**
oppo_lla	0.79	0.54	0.54	0.34	**0.88**
oppo_lua	**0.91**	0.72	0.76	0.48	0.87
oppo_rla	0.76	0.54	0.52	0.28	**0.87**
oppo_rua	0.84	0.67	0.70	0.44	**0.86**
pamap_a	**0.69**	0.68	0.54	0.24	0.27
pamap_c	**0.77**	0.69	0.70	0.06	0.76
pamap_w	**0.83**	0.74	0.63	0.14	0.51

Effectiveness of the Two-Level Index. Now we compare the accuracy and efficiency of using different adaptivity computation strategies including pair-wise comparison, two-level LSH without flattening, and two-level LSH index with flattening. Flattening refers to the step that flattens a set of partitions into a set of JSD-LSH bands, which is the approach that we leveraged for approximating the recursive Jaccard similarity as justified in Sec. 5.2.2. The case where flattening is not applied is equivalent to the case where flattening is applied but the number of JSD-LSH bands L is set to 1. The results are illustrated in Tab. 7. Here, for JSD-LSH, we use $r = 1.4$, number of bands is $L = 200$, number of hash functions is $K = 800$. For Minwise hash, we use 256 Minwise hash functions and use 128 bands. We also allocate 55 samples in each partition, we can see that without the flattening step, the accuracy will significantly drop. In addition, the two-level index achieves about 65 times speedup compared to the pair-wise approach for this scenario.
Effectiveness of JSD-LSH. To justify the use of JSD-LSH, which is a relatively new LSH function and the first LSH function for indexing JS-divergence with a bound, we compare the accuracy and latency of LSH for JS-divergence to a baseline approach that computes the JS-divergences between the target dataset and each of the source datasets directly without using LSH techniques. For

Table 6: Comparison of different similarity measurements for activity recognition

	adaptivity	JS-divergence	l2-distance	source-accuracy	voting
top-1 error	**15%**	31%	23%	92%	38%
top-2 error	**0%**	**0%**	**0%**	92%	30%
top-3 error	**0%**	**0%**	**0%**	77%	23%

Table 7: Comparison of adaptivity computation strategies for activity recognition

	top-1 error	top-2 error	top-3 error	latency (milli-sec)
pair-wise	**15%**	**0%**	**0%**	24, 627
2-level LSH w/o flattening	54%	62%	54%	**377**
2-level LSH w/ flattening	31%	8%	**0%**	**377**

this experiment, we created 162 tables by sampling these 13 activity recognition datasets, so that each table represents samples collected from a body part for a specific subject. The results are illustrated in Fig. 2, in which each bar represents a test case that uses one table as the query (i.e., target dataset), and the rest of the tables as the sources. In this experiment, for each test, we assume the source datasets that have the JS-divergence with the query dataset smaller than 0.1 compose the ground truth. We observe that using LSH can significantly accelerate the overall latency required for JS-divergence comparison, and achieve 5 times speedup on average, while the precision of our proposed approach is 100% and recall is above 93%.

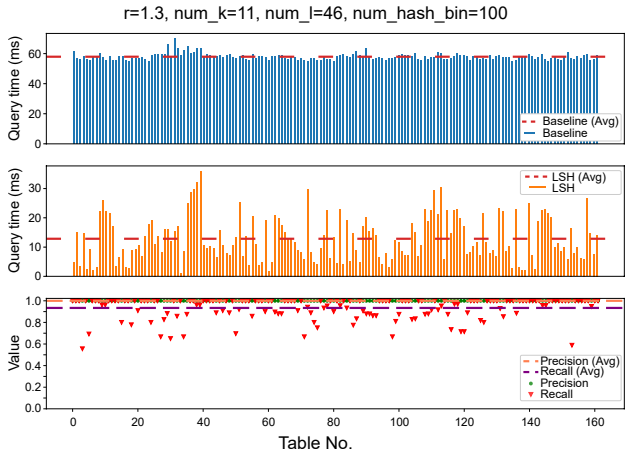


Figure 2: Comparison of LSH for JS-divergence (denoted as LSH) to a pair-wise computation of JS-divergence (denoted as baseline) for 162 tables sampled from 13 activity recognition datasets.

7.3.2 Image Recognition. In this scenario, for each CIFAR-10 image, we normalize all pixel values by dividing each value by 255 and then flatten the image from a tensor of the shape (32, 32, 3) into a one-dimensional vector of the shape (1, 3072). Then, we compute the `adaptivity`, JS-divergence, and L2-distance on the flattened dataset.

Summary. We evaluate and compare the effectiveness of `adaptivity`, JS-divergence, L2-distance, and source accuracy for searching for the best model to serve on a target dataset. The Pearson correlation coefficient of each metric to the target accuracy in each experiment is illustrated in Tab. 8. The top-1 and top-2 error rates for searching for the best model for each experiment are illustrated in Tab. 9. We only show top-1 and top-2 error rates, because we have only four candidate models in this case.

Effectiveness of the Two-Level Index. We also compare the effectiveness of using the two-level index to the pair-wise strategy for computing the adaptivity metric, as illustrated in Tab. 10. For this scenario, each partition has 200 samples. The two-level index is observed to achieve more than 8 times speedup while achieving similar accuracy compared to the pair-wise strategy. The speedup is relatively smaller than the activity recognition scenario. That's mainly because we involve fewer datasets and fewer partitions in this scenario.

Table 8: Comparison of Pearson correlation coefficient for image recognition (the best Pearson correlation coefficient values are highlighted in bold)

target	pcc of adaptivity	pcc of JS-divergence	pcc of l2-distance	pcc of source-accuracy	pcc of voting
Balanced	**0.93**	0.92	0.62	0.80	0.88
Skewed-1	**0.86**	0.53	0.52	0.78	0.48
Skewed-2	0.39	**0.70**	0.66	0.07	0.22
Skewed-3	0.25	0.06	0.18	**0.73**	0.70
Skewed-4	**0.77**	0.41	0.38	0.39	0.64

Table 9: Comparison of error rate for image recognition

	adaptivity	JS-divergence	l2-distance	source-accuracy	voting
top-1 error	**20%**	40%	40%	60%	80%
top-2 error	**0%**	**0%**	**0%**	40%	80%

Table 10: Comparison of adaptivity computation strategies for image recognition

	top-1 error	top-2 error	latency (milli-sec)
pair-wise	20%	**0%**	14, 585
2-level LSH w/ flattening	**0%**	**0%**	**1,652**

Table 11: Comparison of Pearson correlation coefficient (PCC) for text classification (the best Pearson correlation coefficient values are highlighted in bold).

target	pcc of adaptivity	pcc of JS-divergence	pcc of l2-distance	pcc of source-accuracy	pcc of voting
Financial_News_1	0.18	0.49	**0.80**	0.21	0.39
Financial_News_2	**0.93**	0.82	0.92	0.59	0.44
Hate_Tweets	0.78	0.73	**0.79**	0.78	0.70
Tweets_1	0.16	0.24	0.34	0.45	**0.55**
Tweets_2	0.34	0.26	0.23	**0.39**	0.01
IMBD_Movie_1	**0.29**	0.24	0.19	0.06	0.06
IMBD_Movie_2	**0.40**	0.39	0.36	0.11	0.08

7.3.3 Text Classification. In this experiment, as mentioned in Sec. 7.1, each dataset is pre-processed to remove stop words and then tokenized. For each target dataset, a model will be selected from models that are pre-trained on six source datasets. We compare the Pearson correlation coefficient values between the metrics given by measurements of the `adaptivity`, the JS-divergence, the L2 distance, source accuracy, and voting strategy on each source dataset, to the target serving accuracy for the corresponding text classification task. We also compare the top-1 and top-2 error rates.

The results are illustrated in Tab. 11 and Tab. 12, which show that the `adaptivity` measurement has a higher correlation with the target with most of the candidate models as compared to other metrics, however, source accuracy and voting can more effectively predict the best model for serving with least error rate for all seven search scenarios.

In addition, we observed that the `adaptivity` metric can be computed within one minute's time in total for all model search scenarios in this workload, while the voting approach, which is the most time-consuming strategy considered in this paper, takes more than ten minutes. Moreover, the JSD-LSH and the L2 distance are significantly faster than the `adaptivity` measurement, while the source-accuracy approach is the fastest if the source accuracy information has been pre-collected.

7.3.4 Entity Matching. In this experiment, different from activity recognition, most attributes contain text-based values. We represent the training dataset of each entity matching task as a bag of

Table 12: Comparison of error rate for Text Classification

	adaptivity	JS-divergence	l2-distance	source-accuracy	voting
top-1 error	42%	28%	42%	**14%**	**14%**
top-2 error	14%	28%	14%	**0%**	**0%**

Table 13: Comparison of Pearson correlation coefficient of various metrics to the target accuracy for entity matching (the best Pearson correlation coefficient values are highlighted in bold)

target	pcc of adaptivity	pcc of JS-divergence	pcc of source-accuracy
Abt_Buy	**0.99**	**0.99**	**0.99**
Dplp_Acm	0.92	**0.94**	0.70
Dblp_Scholar	0.91	**0.94**	0.83
Walmart_Amazon	**0.99**	**0.99**	0.73
MyAnimeList_AnimePlanet	**0.61**	0.53	0.51
Bikedekho_Bikewale	**0.51**	0.16	0.19
Amazon_Barnes	**0.24**	0.17	0.22
GoodReads_Barnes	**0.61**	0.53	0.20
Barnes_Half	0.29	**0.44**	0.31
RottenTomatoes_IMDB	0.11	**0.34**	0.16
IMDB_TMD	0.49	**0.67**	0.44
IMDB_RottenTomatoes	0.51	**0.75**	0.22
Amazon_RottenTomatoes	0.60	**0.64**	0.38
RogerElbert_IMDB	**0.58**	0.13	0.07
YellowPages_Yelp	**0.82**	0.67	0.44

Table 14: Comparison of error rate for entity matching

	adaptivity	JS-divergence	source-accuracy
top-1 error	**13%**	47%	33%
top-2 error	**0%**	27%	27%

Table 15: Comparison of Pearson correlation coefficient for NLP (the best Pearson correlation coefficient values are highlighted in bold)

target	pcc of adaptivity	pcc of JS-divergence	pcc of source-accuracy
Task1	0.61	**0.71**	0.02
Task2	0.76	**0.87**	0.10

words over a shared dictionary for computing the JS-divergence and `adaptivity`. We compare the Pearson correlation coefficient values of the JS-divergence metric, the `adaptivity` metric, and source accuracy, to the target accuracy for each EM task, as illustrated in Tab. 13. We also compare the overall accuracy in terms of top1-error and top2-error for all fifteen tasks, as illustrated in Tab. 14. The results show that the `adaptivity` metric outperforms other metrics in selecting the models to serve with the best accuracy.

7.3.5 Natural Language Processing. For this experiment, as mentioned in Sec. 7.1, because each task involves merely three datasets, a model discovery scenario for a target dataset only requires comparing two datasets. Therefore, instead of computing the Pearson coefficient for each search scenario, we choose to compute the Pearson coefficient between the metrics and the overall accuracy of each of the two tasks. Each task consists of three different model discovery scenarios. For the same reason, we only consider the top-1 error for this experiment, which is counted over all six model discovery scenarios across the two tasks. The results are illustrated in Tab. 15 and Tab. 16, which show that while `adaptivity` has less correlation with the target accuracy compared to JS-divergence, it can more effectively predict the best model for serving with zero error rate for all six search scenarios. The source accuracy performs significantly worse than other metrics.

Table 16: Comparison of error rate for NLP

	adaptivity	JS-divergence	source-accuracy
top-1 error	**0%**	16.7%	33.3%

7.4 The Impact of Hyperparameter Tuning

We further evaluate how hyperparameters, such as r (as in Eq. 4), the number of concatenated hash functions (i.e., K) of each band, the number of bands (i.e., L), and the number of hash bins for creating probability distribution for comparison, will all affect the accuracy of the JS-divergence computed using LSH compared to the baseline, which are illustrated in Fig. 3. It shows that it is possible to find a set of parameters that ensure high precision and recall by tuning the hyperparameters associated with the LSH scheme.

7.5 The Impact of Partitioning

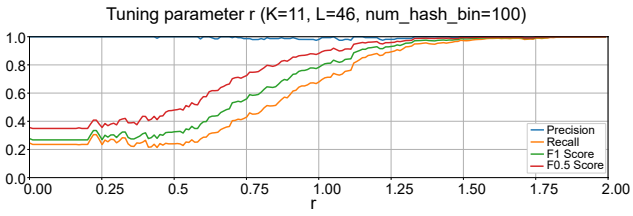
One thing to note is that the computation of `adaptivity` is more complicated than JS-divergence and takes more time. For most of the above experiments, we choose the partition size of each dataset between 300 to 800. For the NLP experiment, when the source dataset and the target dataset have a significant size discrepancy, we choose the size of the smaller dataset to be the partition size, so that the smaller dataset has only one partition. Then we compare the latency of computing JS-divergence and `adaptivity` by using different partition sizes and for different sizes of the source and target datasets. The results are illustrated in Fig. 4, which illustrates that when partition size is around 500, the computational overhead for `adaptivity` is $1.7-3\times$ of JS-divergence. However, the latency of computing `adaptivity` is still $10\times$ lower than the runtime latency of the voting approach which requires each candidate model to run prediction on each sample in the target dataset.

8 RELATED WORKS

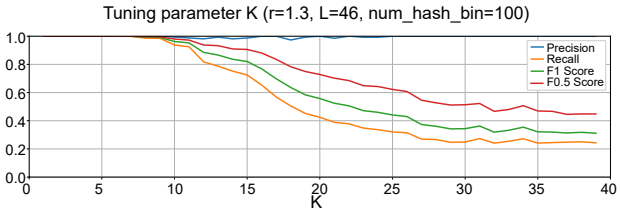
AutoML methods, such as Auto-WEKA [41], Auto-sklearn [18], TPOT [34], H2O [23] automatically search ML model algorithms as well as hyper-parameters. Although their works can also be applied to our targeting problem by modeling the model search process as an optimization problem that minimizes the loss functions defined over the targeting domain, they are mainly designed for the training stage, and their searching strategies require labels of target data for evaluating the loss during the optimization process. In this work, we mainly focus on the deployment stage where it is challenging to collect and manage sufficient labeled data for AutoML tasks.

Clipper [14] proposes a strategy to select models for ensemble inference by modeling the scenario as a multi-armed bandit problem. DJEnsemble [36] presents a cost-based approach for the automatic selection of black-box models to answer spatio-temporal queries. We do not consider such approaches in this work, mainly because they all require labeled data from the target domain for model selection, which could become a significant burden that complicates the model deployment phase.

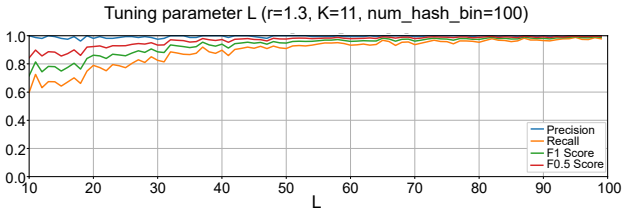
In recent, numerous works are proposed to address open data discovery problems, including automatically discover table unionability [33] and joinability [51, 52], and related tables [48]. Most of these works are trying to identify all similar features using LSH techniques based on Jaccard similarity or variants. While these works can be leveraged to accelerate the match of JSD-LSH signatures, they are not directly applicable in selecting related models,



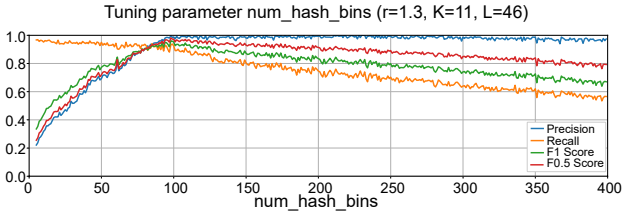
(a) r



(b) number of concatenated hash functions



(c) number of hash tables

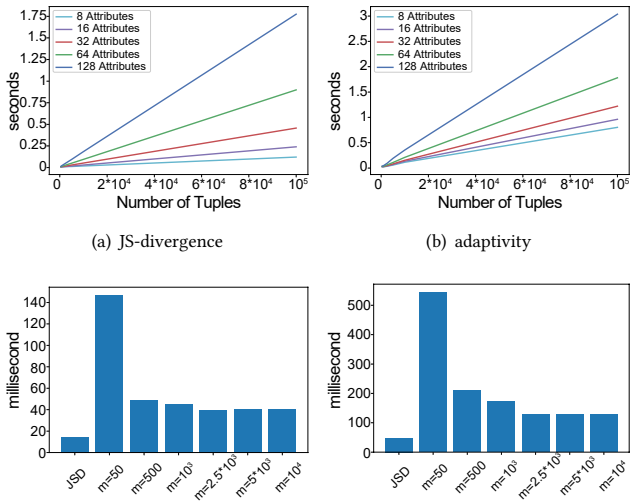


(d) number of hash bins

Figure 3: Hyperparameter tuning for LSH of JS-divergence.

because the existence of a significant portion of shared features between the source and target datasets doesn't mean the probability distributions over the two feature spaces are similar.

Zamir and et al. [47] propose a computational approach to find the transferable relationships, abstracted as taxonomy among the different computation vision tasks, e.g., depth estimation, edge detection, point matching and etc., so that some tasks can be trained using other tasks' output and thus require less training data and supervision budget. Their work is focused on the adaptivity from a task's output domain to another task's input domain. In contrast, our work is focused on the adaptivity of tasks' input domains. Also, while their work is limited to computer vision, our work is targeting a more generalized scenario.



(a) JS-divergence (b) adaptivity

(c) 2500 rows & 32 cols for source/target (d) 10000 rows & 32 cols for source/target

Figure 4: Latency comparison of computing JS-divergence and adaptivity, m is the size of each partition

9 CONCLUSIONS

In a broad class of AI applications, such as Uber's marketplace-level forecasting and downtime prediction for manufacturers, a large number of model versions will be created for different locations and different time points. To facilitate model reuse, we systematically explore the problem of searching models from a repository of pre-trained models that have similar architectures, for serving on a target query dataset. We are particularly interested in solving the problem for fast model deployment scenarios, where little labeled data is available. To address the problem, we propose five techniques, including symmetric similarity-based approaches, an asymmetric similarity-based approach utilizing a novel two-level LSH index based on a new `adaptivity` metric that is not only asymmetric but also convertible into Jaccard similarity, and non-similarity-based approaches. We conducted extensive comparison experiments on a number of workloads including activity recognition, image recognition, text classification, entity matching, and natural language processing. The experimental evaluation showed that our proposed asymmetric similarity-based measurement, adaptivity, outperformed symmetric similarity-based measurements and non-similarity-based measurements in most of the workloads. In the future, we will compare these approaches to other approaches such as AutoML and Clipper, which involve labeled target data.

ACKNOWLEDGMENTS

We would like to thank Devshree Chetan Patel, Ratnam Sanjivkumar Parikh, and Dunchuan Wu, for their help to this work.

REFERENCES

[1] [n. d.]. *Deep Learning Hub.* https://dlhub.app/.
[2] [n. d.]. *Impact of slow page load time on website performance.* https://medium.com/@vikigreen/impact-of-slow-page-load-time-on-website-performance-40d5c9ce568a.
[3] [n. d.]. *PyTorch Hub.* https://pytorch.org/hub/.
[4] [n. d.]. *Tensorflow Hub.* https://www.tensorflow.org/hub.
[5] [n. d.]. *Uber Open Summit 2018.* https://uberopen2018.splashthat.com/.

[6] Akin Avci, Stephan Bosch, Mihai Marin-Perianu, Raluca Marin-Perianu, and Paul Havinga. 2010. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In *23th International conference on architecture of computing systems 2010*. VDE, 1–10.

[7] Billur Barshan and Murat Cihan Yüksek. 2014. Recognizing daily and sports activities in two open source machine learning environments using body-worn sensor units. *Comput. J.* 57, 11 (2014), 1649–1667.

[8] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[9] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1335–1349.

[10] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 380–388.

[11] Ricardo Chavarriaga, Hesam Sagha, Alberto Calatroni, Sundara Tejaswi Digumarti, Gerhard Tröster, José del R Millán, and Daniel Roggen. 2013. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters* 34, 15 (2013), 2033–2042.

[12] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, et al. 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *Proceedings of the fourth international workshop on data management for end-to-end machine learning*. 1–4.

[13] Lin Chen, Hossein Esfandiari, Gang Fu, and Vahab Mirrokni. 2019. Locality-Sensitive Hashing for f-Divergences: Mutual Information Loss and Beyond. In *Advances in Neural Information Processing Systems*. 10044–10054.

[14] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A {Low-Latency} Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.

[15] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n. d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[16] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2017. DeepER–Deep Entity Resolution. *arXiv preprint arXiv:1710.00597* (2017).

[17] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 989–1000.

[18] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074* 24 (2020).

[19] Carlos Roberto González Palacios. 2015. *Optimal Data Distributions in Machine Learning*. Ph. D. Dissertation. California Institute of Technology.

[20] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[21] Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J Gao. 2019. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *Proceedings of the VLDB Endowment* 12, 7 (2019), 822–835.

[22] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.

[23] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020.

[24] Lillian Lee. 2000. Measures of distributional similarity. *arXiv preprint cs/0001012* (2000).

[25] Joseph Lee Rodgers and W Alan Nicewander. 1988. Thirteen ways to look at the correlation coefficient. *The American Statistician* 42, 1 (1988), 59–66.

[26] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584* (2020).

[27] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151.

[28] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2346–2363.

[29] Bill MacCartney and Christopher D Manning. 2009. *Natural language inference*. Citeseer.

[30] Xian-Ling Mao, Bo-Si Feng, Yi-Jing Hao, Liqiang Nie, Heyan Huang, and Guihua Wen. 2017. S2JSD-LSH: A locality-sensitive hashing schema for probability distributions. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 3244–3251.

[31] Hui Miao, Ang Li, Larry S Davis, and Amol Deshpande. 2017. Modelhub: Deep learning lifecycle management. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1393–1394.

[32] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[33] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[34] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.

[35] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[36] Rafael Pereira, Yania Souto, Anderson Chaves, Rocio Zorilla, Brian Tsan, Florin Rusu, Eduardo Ogasawara, Artur Ziviani, and Fabio Porto. 2021. DJEnsemble: a Cost-Based Selection and Allocation of a Disjoint Ensemble of Spatio-temporal Models. In *33rd International Conference on Scientific and Statistical Database Management*. 226–231.

[37] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*. IEEE, 108–109.

[38] Sebastian Schelter, Joos-Hendrik Boese, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2017. Automatically tracking metadata and provenance of machine learning experiments. In *Machine Learning Systems Workshop at NIPS*. 27–29.

[39] Anshumali Shrivastava and Ping Li. 2015. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th international conference on world wide web*. 981–991.

[40] Chong Sun, Nader Azari, and Chintan Turakhia. 2020. Gallery: A Machine Learning Model Management System at Uber.. In *EDBT*. 474–485.

[41] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 847–855.

[42] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–3.

[43] Jindong Wang, Yiqiang Chen, Lisha Hu, Xiaohui Peng, and S Yu Philip. 2018. Stratified transfer learning for cross-domain activity recognition. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 1–10.

[44] Christian Weber, Pascal Hirmer, and Peter Reimann. 2020. A Model Management Platform for Industry 4.0–Enabling Management of Machine Learning Models in Manufacturing Environments. In *International Conference on Business Information Systems*. Springer, 403–417.

[45] Yang Yang, De-Chuan Zhan, Ying Fan, Yuan Jiang, and Zhi-Hua Zhou. 2017. Deep Learning for Fixed Model Reuse.. In *AAAI*. 2831–2837.

[46] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, and Chris Jermaine. 2020. Tensor Relational Algebra for Machine Learning System Design. *arXiv preprint arXiv:2009.00524* (2020).

[47] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. 2018. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3712–3722.

[48] Yi Zhang and Zachary G Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1951–1966.

[49] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *The World Wide Web Conference*. 2413–2424.

[50] Lixi Zhou, Jiaqing Chen, Amitabh Das, Hong Min, Lei Yu, Ming Zhao, and Jia Zou. 2022. Serving Deep Learning Models with Deduplication from Relational Databases. *arXiv preprint arXiv:2201.10442* (2022).

[51] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.

[52] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH ensemble: Internet-scale domain search. *arXiv preprint arXiv:1603.07410* (2016).

[53] Jia Zou, Pratik Barhate, Amitabh Das, Arun Iyengar, Binhang Yuan, Dimitrije Jankov, and Chris Jermaine. 2020. Lachesis: Automated generation of persistent partitionings for big data applications. (2020).

[54] Jia Zou, Arun Iyengar, and Chris Jermaine. 2019. Pangea: monolithic distributed storage for data analytics. *Proceedings of the VLDB Endowment* 12, 6 (2019), 681–694.