# Towards an End-to-End Training Data Management System for Machine Learning Models

by

Huayi Zhang

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Data Science

by

---

January 2023

APPROVED:

| | |
|---|---|
| Elke A. Rundensteiner<br>Worcester Polytechnic Institute<br>Advisor | Oren Mangoubi<br>Assistant Professor, WPI.<br>Committee Member |
| Xiaozhong Liu<br>Associate Professor, WPI.<br>Committee Member | Samuel Madden<br>Professor, MIT<br>External Committee Member |

## Towards an End-to-End Training Data Management System for Machine Learning Models


### Abstract

A successful machine learning application requires powerful machine learning models and high-quality training data. In recent years, the machine learning community has proposed many powerful deep learning models that achieve promising results on benchmark machine learning tasks. However, techniques and tools to help practitioners to prepare and evaluate high-quality training data remain relatively limited. The goal of this dissertation is thus to propose an end-to-end training data management system that helps practitioners on the following tasks detailed below.

- **Removing Anomalies from the Raw Dataset:** 1. Raw datasets collected in real applications may contain low-quality data samples, such as anomalies produced due to sensor issues. To tackle this, I develop a semi-supervised anomaly detection algorithm called ELITE. While only requiring the users to label an extremely small set of samples, ELITE significantly improves the state-of-the-art deep anomaly detection models. Our experiments on public benchmark datasets show that ELITE achieves up to 30% improvement in ROC-AUC score compared to the state-of-the-art techniques. This project has been published in KDD2021.
- **Labeling Training Data:** In practice, it is often infeasible to manually label a sufficient number of training samples for modern large-scale machine learning models. To minimize labeling efforts by domain experts, I propose a label propagation system, LANCET, that automatically propagates manually annotated labels to similar unlabeled data objects. LANCET addresses three challenges in an integrated framework: (1) which data samples to ask humans to label, (2) how to propagate labels to other samples automatically, and (3) when to stop labeling. Our experiments on diverse public data sets demonstrate that LANCET consistently outperforms state-of-the-art methods by a large margin – up to 30 percentage increase in accuracy. This project has been published in VLDB2021.
- **Curating Training Data:** The training data preparation tools, such as the ones mentioned above, still inevitably generate erroneous training data. To tackle this, I develop the MetaStore system to help data scientists curate deep learning models' training data based on training samples' gradients produced during the model training process. MetaStore supports efficient gradient-based analytics query execution with three key components : (1) a lightweight gradient collector, (2) a compact gradient storage, and (3) an efficient gradient analytics engine. Our experiments demonstrate that MetaStore outperforms alternative baseline methods from 4 to 578x in storage costs and from 2 to 1000x in running time. This project has been submitted to VLDB2023.

# Contents

**Acknowledgements**
I would first like to thank my advisor Professor Elke A. Rundensteiner, who provided me the opportunity to work on this exciting project. Her guidance helped me in all the time of research and writing of this dissertation.

My sincere thanks also go to Professor. Lei Cao at the University of Arizona for his patience, motivation, and immense knowledge. I can't imagine finishing this topic without his guidance. I would also like to thank student colleague Yizhou Yan, Peter VanNostrand, Dennis Hofmann at the WPI DAISY Lab for their help.

I would also like to thank Professor Samuel Madden,, Professor Xiaozhong Liu, Professor Oren Mangoubi for serving as the reader of this paper. I am gratefully indebted to them for their very valuable comments on this dissertation.

Finally, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

# 1 Introduction

In recent years, the rapidly growing machine learning community has proposed many powerful machine learning models. These modern models, especially deep learning models, achieve state-of-the-art results in various domains. For example, in the computer vision area, the EfficientNet [99] model achieves 90.2% accuracy on the challenging imagenet dataset. The RoBERTa [65] model obtains 97.5% accuracy on the SST semantic classification task in the Natural Language Processing area. These exciting accomplishments raise significant interest in applying these novel deep learning models to real-world applications, such as digital healthcare and autonomous vehicles.

However, the lack of high-quality training data continues to represent a critical bottleneck. Despite their strong performance, most existing deep learning models must be trained on carefully annotated and error-free benchmark datasets, while the real-world data arising in practice are often unlabeled and noisy. In order to utilize the powerful machine learning models, it usually requires practitioners to prepare enough high-quality training data. In fact, [70] shows that it takes around 40% of project time to collect and prepare training datasets.

With the development of big data techniques, machine learning practitioners are usually able to collect and store a large amount of data in real-world applications. However, these raw data samples can rarely be directly used as the training data of machine learning models. To prepare a high quality training dataset, the practitioners commonly face two challenges, 1) efficiently identifying the erroneous data samples (such as the outliers produced due to hardware issues) and 2) accurately annotating the raw data samples. Furthermore, instead of preparing the training dataset as a one-shot pre-training step, the machine learning practitioners often need to adjust the training dataset during the model development process. After training a few models with the existing training dataset, the practitioners can diagnose the trained models and determine if there are mislabeled samples that need to be removed or if some additional samples needs to be collected to improve the model performance. [6]

This dissertation thus aims to propose an end-to-end system that assists practitioners with three critical tasks inherent in this training data construction process, namely, **Cleaning, Labeling, and Curating** training data.

## 1.1 Motivation: The Need for an End-to-End Training Data Management System

As a motivating example, I have been participating in a large-scale training data management system project led by researchers at MIt and at the Neurology Department of Massachusetts General Hospital, which is used to prepare a training dataset with EEG segments (450 million segments, 30TB data size) with six classes representing different types of seizures. This training dataset is used to train a seizure classifier that automatically detects seizures based on EEG signals collected during the clinical observation of

patients. We conjecture that a successful system should tackle three key tasks to construct a high quality dataset for training an accurate deep learning model.

**Erroneous Data Removal:** Our neurologist collaborators observe that the raw dataset contains a small ratio of erroneous EEG segments that are produced due to hardware problems, such as broken or improperly installed sensors. These low-quality segments are useless for the classification task and potentially stall the model's training process. Such erroneous EEG segments thus should be identified and removed from the raw dataset before labeling the data.

**Data Labeling:** Our neurologist collaborators expect that well over 20 million labeled EEG segments are needed to cover the full range of variations in seizures. Relying on domain experts to manually provide this large quantity of labels is impractical, given medical experts' time is extremely limited. An ideal system should help neurologists to label these segments with minimal annotation efforts.

**Training Data Curation:** After a training dataset has been annotated by the neurologist, the machine learning experts in our team train many deep learning models with different model architectures and hyper-parameters. However, they find that the downstream models cannot meet the expected accuracy. They suspect this is because there are mislabeled segments in the given training dataset. They hope the system could help them identify some potential mislabeled samples for our neurologist collaborators to review.

## 1.2 ELITE: Robust Deep Anomaly Detection with Meta-Gradient

One key challenge as well as opportunity in preparing a training dataset is to remove the low quality data objects, such as the anomalies produced due to measuring hardware issues. One solution is to regard this problem as a deep anomaly detection task. Deep anomaly detection methods typically attempt to learn the distribution of normal samples. Then, any sample that is out of the distribution is rejected as an anomaly.

### 1.2.1 State-of-the-Art in Deep Anomaly Detection

In recent years deep neural networks have been widely used to detect anomalies from complex data sources, such as imagery and time series [46, 82, 40]. Because real applications typically do not have a large number of labeled anomalies available beforehand, most deep anomaly detection techniques are either unsupervised [46, 82, 40] and do not use any labels or semi-supervised [82, 40, 124] and use a small set of normal or abnormal examples to improve the accuracy of unsupervised deep anomaly techniques.

However, these deep anomaly methods, either unsupervised or semi-supervised, require that the unlabeled training data be clean – not contaminated by any anomalies – so that they can learn a data representation that captures the distribution of the normal data. Were the training data to be contaminated by anomalies, the representation learned by these deep models could encode information about anomalous samples as part of the distribution of normal data. In this case, there is no guarantee that these models can properly

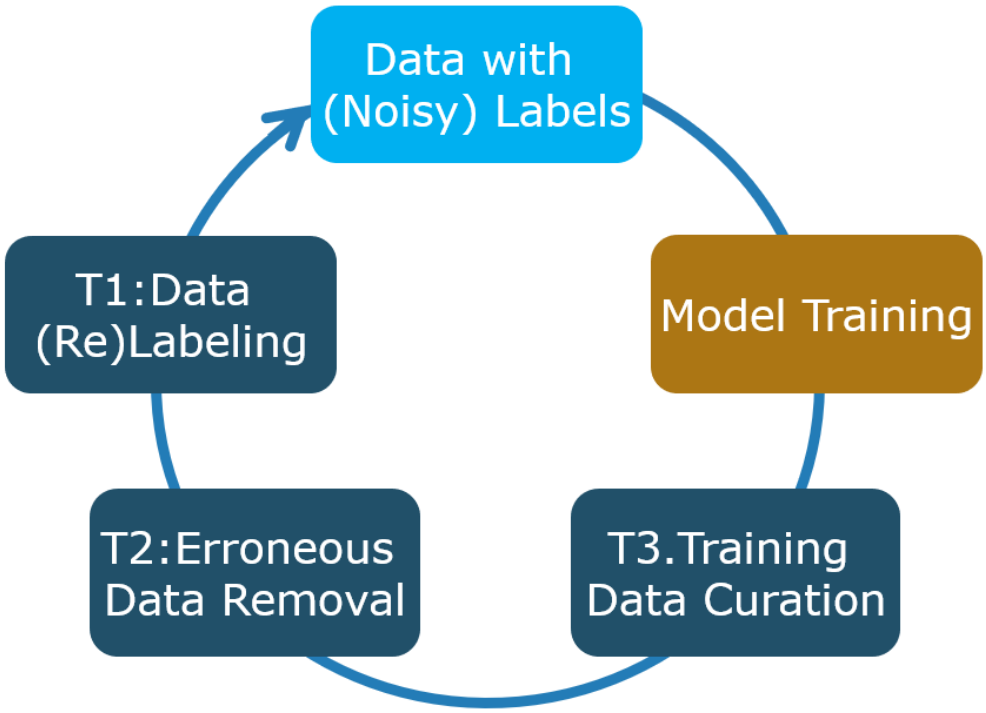



Figure 1: This dissertation focuses on three key tasks in the training data construction process, **(Task 1). Cleaning and (Task 2.) Labeling** training data before building the model, and **(Task 3.) Curating** training data during the model training process.

distinguish between normal and anomalous samples. Unfortunately, in real applications, such a clean training data set rarely exists.

Although semi-supervised deep anomaly methods improve the quality of unsupervised anomaly detection by leveraging the classical semi-supervised classification strategy, they still suffer from polluted training data. As shown in our experiments (Sec. 5.2), their performance degrades quickly when the number of the anomalies in the training data increases.

### 1.2.2 Proposed Task I: ELITE: Robust Deep Anomaly Detection with Meta-Gradient

In this part, I propose an approach, called ELITE, that leverages labeled examples to solve the problem caused by polluted training data. Unlike the semi-supervised classification strategy which uses labeled examples as training data, ELITE uses the labeled examples as *validation data*. The core methodology of ELITE is to infer the labels of the polluted training data samples as normal or anomalous according to their potential influence on the model's validation loss. ELITE is based on a basic hypothesis: the correct labels of the unlabeled training samples should reduce the validation loss on the labeled examples.

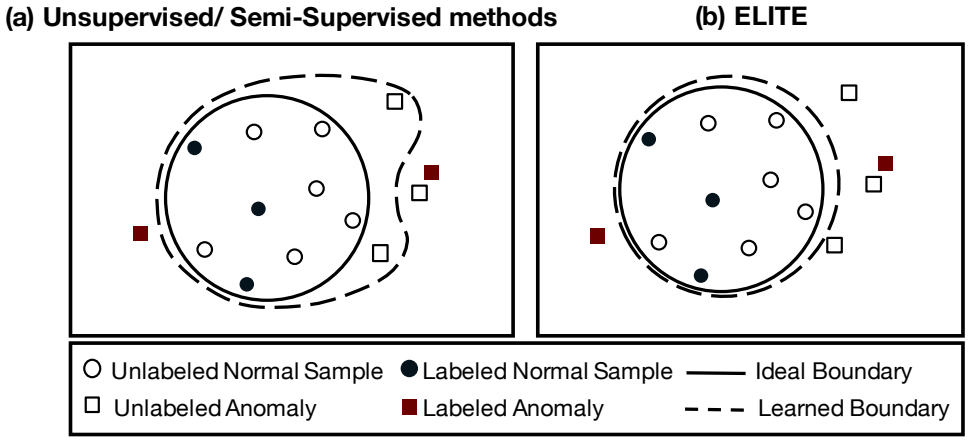



Figure 2: ELITE: Robust to Polluted Training Data. Leveraging the labeled examples, ELITE turns the hidden anomalies into useful signals that help to learn a better classification boundary.

Thus ELITE uses a strategy that continuously discovers anomalies in the polluted training data and learns a better deep anomaly model based on the corrected labels.

Moreover, using a tailored loss function that copes with normal and anomalous samples differently, ELITE trains the model to *maximize* the anomalous score for unlabeled samples that are likely anomalies while *minimizing* this score for unlabeled samples that are likely normal. In this way, ELITE not only uses the information from labeled examples, but also effectively turns the anomalies in the training data into *useful signals* that help to produce a data representation inherently anomaly-aware.

Clearly, the key of ELITE is how to efficiently identify the optimal labels for the unlabeled samples that minimize the model's validation loss. Finding optimal labels by repeatedly flipping the label of each sample and re-training the model to compute the validation loss will be too expensive. To solve this problem, ELITE proposes an efficient label inference method, called ALICE. ALICE introduces the concept of *meta-gradients* to directly estimate the potential change of the validation loss caused by altering the label of any training sample, without having to re-train the model. ELITE then fuses ALICE into every iteration during the training process to dynamically adjust the labels of the training samples in a way that is guaranteed to *monotonically* reduce the validation loss. ELITE is general in that different categories of unsupervised deep anomaly techniques can seamlessly plug their objective functions into ELITE and benefit from the labeled examples, such as Auto-Encoder-based methods [42, 84, 7, 122, 18] and Deep One Class Classification-based methods [86, 81], as discussed in Sec. 3.4 and confirmed by our experiments (Sec. 5).

## 1.3 LANCET: Labeling Complex Data At Scale

In this dissertation, I tackle the task of labeling complex data task by building a labeling system that produces a sufficient number of labels with minimal human effort. To achieve this, several problems have to be solved:

(1) *Which objects to ask human experts to label?* To minimize the labeling efforts, a subset of objects must be selected for the human annotators to label. Compared to other possible subsets of objects, labeling of this chosen subset should *maximally* improve the performance of the machine learning model subsequently trained on it. This selection is critical for minimizing the human effort, especially when annotating data requires strong domain knowledge.

(2) *How to automatically generate additional labels?* Given that human experts will not be able to provide all necessary labels, a mechanism is needed to auto-generate labels by propagating manually produced labels to similar but still unlabeled objects. This label propagation is difficult, particularly when dealing with complex data such as EEG/EKG data signals, video clips from autonomous vehicles, or other complex data types. First, when measuring the similarity of complex and often high-dimensional objects, we cannot simply rely on raw features of the data to distinguish between objects of different classes. Second, we need an appropriate distance function to measure the closeness of pairs of objects. Third, a distance threshold is required to determine if two objects are close enough to be labeled by the same class.

(3) *When to stop the labeling process?* To avoid wasting valuable human labeling effort, an effective labeling system must determine whether the labels acquired so far are sufficient to achieve high training accuracy of the machine learning model. Although acquiring more labels may not hurt the performance of the machine learning model, the performance improvement brought by new labels is expected to diminish as the number of the labels increases. Therefore, an effective labeling system should automatically determine when the promise of additional performance improvement diminishes sufficiently so to safely terminate the labeling process.

### 1.3.1 State-of-the-Art in Labeling Training Data

Although some previous research has introduced techniques for reducing human labeling efforts, in particular by weak supervision [76, 103, 24] and active learning [30, 31, 28, 94, 88, 109, 94], none of these works solve all three problems outlined above.

First, weak supervision methods such as Snorkel [76], Snuba [103], and GOGGLES [24] automatically generate labels using some labeling seeds. However, they assume the labeling seeds are provided either via some user-defined labeling functions or are a priori user-supplied labeled examples. Although these labeling seeds impact the number and the error rate of the automatically produced labels, weak supervision does not address this critical question of which seeds are most effective at producing quality labels and thus should be initially acquired. That is, weak supervision addresses Problem 2 (Automatic Label Generation), but does not solve Problem 1 (Label Candidate Selection).

The problem of label candidate selection (Problem 1) is the focus of active learning [30, 31, 28, 94, 88, 109, 94, 101]. Active learning, however, does not tackle the challenge of automatically generating labels (Problem 2). Further, active learning techniques were typically designed to serve specific machine learning models such as Convolutional Neural Networks [88] or Support Vector Machines [101]. Even if the labeling candidates recommended by active learning were potentially effective at improving the accuracy of specific prediction models, they are not guaranteed to be effective when used for automatic label generation.

### 1.3.2 Proposed Task II: LANCET: Labeling Complex Data at Scale

In this work, I propose the first end-to-end solution called LANCET, that effectively that tackles three research problems: *1) Which objects to ask human experts to label? 2) How to automatically generate additional labels? 3) When to stop the labeling process?*

Instead of providing a collection of independent techniques each targeting only one of the three research problems in isolation, LANCET solves all three of the above core automated labeling problems using a principled approach.

LANCET is built on a theoretical foundation that introduces the Covariate-shift condition and the Continuity condition in Sec. 7, guiding me to develop the label propagation and label candidate selection strategies.

**Label Propagation.** The Covariate-shift condition means that the unlabeled objects share the label with their near-by labeled neighbors. Intuitively if the distributions of the labeled objects and the unlabeled objects satisfy the Covariate-shift condition, then I can accurately produce labels by automatically propagating labels from the labeled objects to their near-by unlabeled neighbors. However, this Covariate-shift condition rarely holds true in the raw feature space of complex data. To solve this problem, I propose a feature embedding strategy, called *conditional feature matching* (CFM), to learn a new feature space satisfying this Covariate-shift condition. The key idea is to express the Covariate-shift condition as a *conditional feature matching loss* that serves as a regularization term to the loss function of any semi-supervised feature embedding method I plug into LANCET.

I then show in Sec. 9, that in this feature embedding space, a simple linear model would be sufficient to automatically propagate labels – without having to explicitly specify a distance function nor a distance threshold. This is beneficial as it is often impossible for experts familiar with their domain but not with machine learning to provide such machine learning specific parameters and functions.

**Label Candidate Selection.** Next, I observe that to reliably assign a label to each unlabeled object in a to-be-labeled dataset, each unlabeled object should always have some sufficiently near-by labeled neighbors in this embedding space. This observation is formalized as the Continuity condition in Sec. 7.

Guided by the Continuity condition, I design a *label candidate selection* strategy. It selects the candidate objects to be labeled by the domain experts as those whose coordinates $x$ in the data space have a large value on the probability density function (PDF) of the un-

labeled objects, but a small value on the PDF of the labeled objects. I call these unlabeled objects as the objects with large *probability density ratios*. These objects are not well represented by existing labeled objects.

Because estimating the PDF is notoriously hard in a high dimensional space [91, 119, 22, 27], I propose a strategy that estimates the *density ratios* without having to explicitly know the PDF. It leverages our insight that the density ratio estimation problem can be mapped to a distribution matching problem. That is, I show that given a set of weights w.r.t. each unlabeled object, if these weights together minimize the difference (distance) between the *weighted distribution* of the unlabeled objects and the distribution of labeled objects, they effectively estimate the density ratios.

I then solve this distribution matching problem by designing a *distribution matching network* (DMN), which given an object, determines if it is sampled from the weighted unlabeled distribution or the labeled distribution. I then learn these weights by *maximizing* its classification error rate and thus effectively minimizing the difference between those two distributions.

**Labeling Termination.** Using the above DMN, I establish an *effective termination condition* for the labeling process. Namely, given objects in our data set, if the classification error rate of the DMN is close to 0.5 – indicating that it fails to distinguish labeled objects from unlabeled objects – then the labeling process should stop. The intuition is that because this condition assures that labeled objects effectively represent the distribution of the unlabeled objects, it would no longer improve the accuracy of the prediction model if I were to label more objects.

The *conditional feature matching* strategy, the *label candidate selection strategy*, and the labeling termination condition jointly ensure that LANCET produces a sufficient number of quality labels with minimal human efforts.

## 1.4 MetaStore: Meta Data Analytics for Training Data Curation

The training process of DNNs produces a massive amount of meta-data, including feature embeddings [104], losses [89], and gradients [77]. This meta-data, if analyzed appropriately, contains significant value that can be leveraged for a number of tasks critical to achieving superior model performance. These tasks include but are not limited to cleaning noise in the training data, explaining the behavior of trained models, or reusing and fine-tuning models. For example, research [14, 74, 104] has shown that analyzing training loss and feature embeddings can explain inference results and help debug DNN models.

To address this need, we are developing a system called **MetaStore**, that collects, stores, and then analyzes such meta-data artifacts at scale. It consists of three key components: meta-data collection, meta-data storage, and meta-data analytics. During deep learning modelling, the meta-data collection component non-obtrusively collects a variety of meta-data types, which the meta-data storage engine maintains. The meta-data analytics engine then allows users to conduct high-performance analytics over the meta-data store.

**Promise of Gradient Meta Data.** In this dissertation, we focus on one particular type of meta-data: gradients. DNNs commonly train models using a sequence of *gradient descent* steps which gradually fit the model parameters to the training data. Thus, as the bridge between the data and the model, these gradients can be used to effectively estimate the influence of training samples or hyper-parameters on the learned model parameters. For example, in the machine learning literature, many robust deep learning techniques use *gradients* [77, 127, 114, 50, 13] during DNN training for a range of tasks, including mitigating the impact of potential noise in the training examples and dynamically adjusting the hyper-parameters such as learning rate.

Similarly, in our setting of offline meta-data analytics, as discussed in Sec. 14.2 and as shown in our experiments, if these gradients can be compactly stored and appropriately analyzed, they are of great value in discovering mislabeled training data and anomalies, explaining model inference results by the relevant training samples, and guiding the collection of new training data to improve the performance of the model. However, to date no tools have been developed that effectively collect, store, or analyze gradients, because the size of the gradient tends to be huge.

### 1.4.1 Performance Challenges.

DNNs are typically is composed of many layers, including convolutional layers, linear layers, batch normalization, etc. A DNN computes gradients with respect to the training examples layer-by-layer. At each layer, the dimensionality of the gradient is equivalent to the number of parameters in the layer; many modern DNN models are huge, with up to billions of parameters [26]; indeed, this size is essential for achieving superior performance in modern ML tasks.

As an example, in the well-known DNN models, such as ResNet [44] or VGG [93], a single linear layer can have $4096 \times 4096$ parameters. Given a CIFAR-10 dataset with 50,000 training samples – in itself not considered a very large data set, it would take about **20.9 TB of disk space** to store the gradients produced in just one *single* linear layer. Worst yet, deep learning trains models epoch by epoch and thus produces this amount of gradients per epoch.

Therefore, merely **storing** this volume of gradients w.r.t. one (or worse yet all) layers quickly becomes infeasible. Even if we had sufficient (near infinite) storage resources for keeping all such gradient data for each round of training, the mere task of just **collecting** these gradients would be challenging itself. Directly logging the gradients produced by the DNN training in an online process would dramatically slow down the already exceedingly expensive training process. Moreover, loading this huge amount of meta-data into memory for *analytics* would introduce *exorbitant I/O costs* during query execution. On the other hand, if we instead were to re-compute the gradients on-the-fly whenever a gradient analytics query is issued, this would cause *prohibitive query execution costs*. This is because computing a gradient from scratch suffers from a time complexity quadratic in the number of parameters, and effectively requires re-execution of the NN training

pipeline.

### 1.4.2 Proposed Task III: MetaStore: Meta Data Analytics for Training Data Curation

By exploiting the properties of popular DNN models and their gradient computation methodology MetaStore is able to offer an effective solution to these challenges.

***MetaStore Compact Data Storage.*** First, our careful analysis of the back-propagation process of DNN training reveals that the huge gradient of a training sample can be decomposed into 2 small gradients, namely, *prefix* and *suffix* gradients, from which the gradient can be *exactly* re-constructed via a matrix product operation. These two partial-gradients are typically several orders of magnitude smaller than the original gradient especially when produced in layers with a huge number of parameters. Therefore, they are extremely compact, cutting the storage costs from terabytes to gigabytes.

***MetaStore Lightweight Data Collection.*** Instead of first computing the full gradient and then manually decomposing it, we observe that both the small prefix and suffix gradients correspond to intermediate data that could naturally be produced during the back-propagation step when computing the gradient. Their collection can thus be done almost for free, i.e., via a very lightweight process. Better yet, this process is backwards compatible with existing learning processes. For this reason, it can be easily integrated into standard deep learning frameworks, such as PyTorch or Tensorflow, without requiring any system modification. This in turn will improve MetaStore's ease of adoption in practice.

***MetaStore Efficient Analytics.*** MetaStore is the first system to provide *a rich set of operators* that allow users to conduct many gradient-based analytics on the stored metadata from discovering erroneous training samples to interpreting model behavior. These operators often involve computing the inner product similarity of two gradients. This inner product operation tends to impose a significant computational bottleneck [114, 127]. Worst yet, if we were to store two separate prefix and suffix gradients in place of the actual gradient, this would further slow down the inner product operation because MetaStore would have to reconstruct the original gradients each time, before proceeding to execute the specific analytics operations.

In this work, we design an efficient strategy that is able to exactly compute the gradient inner product *without first reconstructing the gradients*. It leverages our observation that given two gradients in a *linear layer*, we can directly use their respective prefix and suffix gradients to compute their inner product via a lightweight linear algebra transformation. With the prefix and suffix gradients much smaller than the gradient itself, this speeds up the inner product operation by several orders of magnitude.

***Generality of the MetaStore Approach.*** The efficient collection, storage, and analytics services of MetaStore are applicable to all common types of layers (beyond just linear layers) found in popular DNN models such as ResNet [44], VGG [93], and BERT [26]. This holds because all the commonly used layers in these models (each potentially with many trainable parameters and thus producing large gradients), such as the convolutional lay-

ers and the self-attention layers, can be decomposed into a set of linear layers.

## 1.5 List of Proposed Dissertation Tasks & Road Map

In this dissertation, I propose to complete the following tasks to build an end-to-end training data debugging system to address the challenges laid out above. The dissertation is organized into three parts. Part I develops solution that addresses the need for removing anomalies in the raw dataset. Part II then investigates how to minimize the human effort needed to label a large amount of training data. Finally, part III aims to build a training data debugging system that identifies erroneous training data objects during the model development process. The road map of these three tasks is detailed as below,

1. Part I has been accomplished, resulting in the research paper: Huayi Zhang, Lei Cao, Peter VanNostrand, Sam Madden, and Elke Rundensteiner. "ELITE: Robust Deep Anomaly Detection with Meta Gradient", ACM SIGKDD 2021.

2. Part II has been accomplished, resulting in the research paper: Huayi Zhang, Lei Cao, Samuel Madden, Elke Rundensteiner. "LANCET: Labeling Complex Data at Scale", VLDB 2021.

3. Part III has been accomplished, resulting a research paper "MetaStore: Meta Data Analytics at Scale" that is in submission to SIGMOD2023.

# Part I
# ELITE: Outlier Removal From Training Data

(The method described in this part has been accepted at KDD2021.)

## 2 Preliminaries

### 2.1 Problem Definition

Given a set of unlabeled training samples $\mathbb{X}^U : \{x_1^u, \cdots, x_N^u\}$ that contains anomalies, and a small set of labeled samples $\mathbb{X}^L : \{(x_1^l, y_1^l), \cdots, (x_M^l, y_M^l)\} \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{Y} \in \{-1, 1\}$ with $y^l = 1$ denoting normal sample and $y^l = -1$ denoting anomalies, the goal is to train a neural network $\phi(x; \theta)$ that assigns small anomalous scores to normal data and large anomalous scores to anomalies:

$$\Omega(x)|_{y=-1} \geq \Omega(x)|_{y=1} + C \tag{1}$$

In Eq. 1, $\Omega(x)$ represents the anomalous score of $x$, while $C$ is a hyper-parameter that controls the margin of anomalous score between normal data samples and anomalies.

### 2.2 Unsupervised and Semi-supervised Deep Anomaly Detection

To better present our proposed approach in Sec. 3, in this section we briefly introduce the key concepts of unsupervised and semi-supervised deep anomaly detection, using one-class classification-based methods [86, 81], deep Auto-Encoder-based methods [42, 84, 7, 122, 18], and semi-supervised DeepSAD [82] as examples.

#### 2.2.1 Unsupervised Deep Anomaly Detection

Let $\phi(x; \theta)$ be a neural network parameterized by $\theta$, and $\Omega(x)$ be the anomalous score function for a data sample $x$. The goal of deep one-class classification [86, 81] is to map the training samples into a compact hypersphere in the learned latent space, where $\Omega(x) = \|\phi(x, \theta) - o\|^2$ with $o$ denoting the center of the learned hypersphere.

The Auto-Encoder-based methods train a dimension reduction model that reconstructs all training samples with small error. Naturally, it uses the reconstruction error as the anomalous score function, i.e. $\Omega(x) = \|\phi(x; \theta) - x\|$. The training objective is to minimize the average anomalous score of the training samples as shown in Eq. 2.

$$\arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \Omega(x) \tag{2}$$

These unsupervised deep anomaly methods work well when the training dataset contains no or only very few anomalies. However, this assumption does not hold in many real applications. Minimizing the anomalous score of all training samples thus causes performance degradation as discussed in Sec. 1.

### 2.2.2 Semi-Supervised Deep Anomaly Detection

As a semi-supervised deep anomaly method, DeepSAD [82] uses the training loss incurred on the labeled anomaly samples to compensate the loss function of the unsupervised Deep SVDD [81].

$$\arg\min_{\theta} \frac{1}{N+M} \sum_{i=1}^{N} \|\phi(x_i, \theta) - o\|^2 + \frac{1}{N+M} \sum_{j=1}^{M} (\|\phi(x_j, \theta) - o\|^2)^{y_j} \tag{3}$$

In Eq. 3, $o$ represents a vector in the deep feature embedding. $N$ and $M$ are the size of the unlabeled and labeled set respectively. The first part of Eq. 3 is identical to the loss function of the unsupervised Deep SVDD [81]. We call it *unsupervised loss*. The second part corresponds to the *supervised loss*. As a penalization function, it pushes the labeled anomalies further away from the center.

# 3 Proposed Method: ELITE

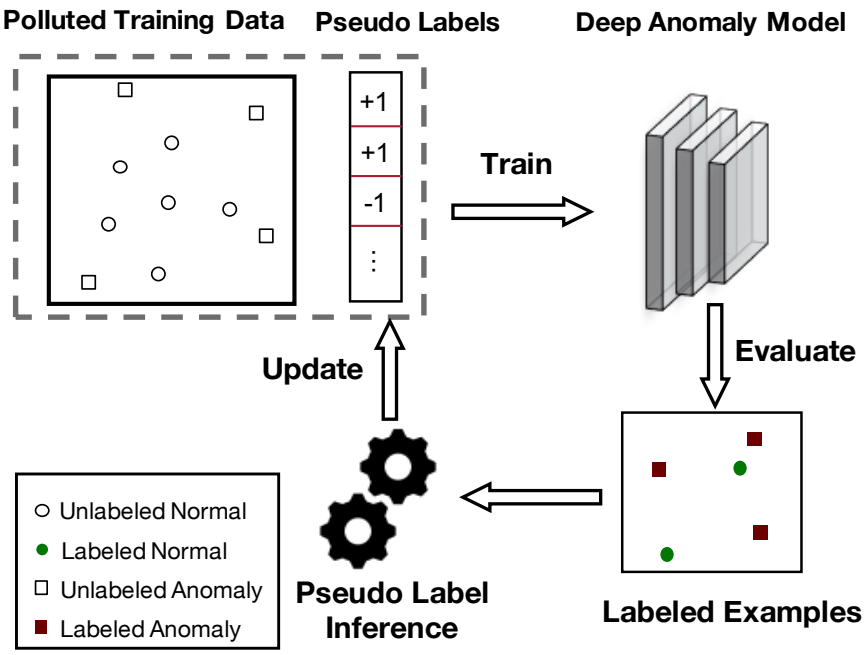



Figure 3: Overall Process of ELITE

## 3.1 Overall Process of ELITE

Next, we introduce ELITE, a novel approach that effectively leverages a small number of labeled examples to solve the pollution problem of training samples. ELITE uses the labeled examples as validation set to evaluate the model trained on the unlabeled training samples. The key idea of ELITE is to infer the labels of the unlabeled training samples as normal or anomalous according to the potential influence on model's validation loss. It then learns from the corrected labels a better deep anomaly model. In this way, ELITE no longer relies on the availability of a clean training dataset.

Fig. 3 depicts the overall process of ELITE. Given a polluted training set $\mathbb{X}^U : \{x_1^u, \cdots, x_n^u\}$, ELITE starts with assigning a *pseudo label* to each sample in $\mathbb{X}^U$ and trains a deep learning model on these pseudo labels. Initially, we assume all samples are normal. It then uses the labeled examples to validate the effectiveness of the model. Next, ELITE uses a pseudo label inference method that leverages the gradient of the validation loss to correct the pseudo labels of the training samples in a way guaranteed to reduce the validation loss. ELITE then updates the deep anomaly model based on the corrected labels. It iterates the pseudo label inference and model update steps until updating the labels of the training samples no longer decreases the validation loss. ELITE deploys the final deep anomaly model to detect anomalies from user data.

In the rest of this section, we first introduce ELITE's objective functions including the training loss and validation loss in Sec. 3.2. Then in Sec. 3.3 we propose an effective strategy to update the pseudo labels and analyze its time complexity and convergence. Finally, we show how ELITE works seamlessly with the existing unsupervised anomaly methods using Deep SVDD [81] as example.

## 3.2 Objective Functions

### 3.2.1 Training loss

The objective of ELITE is to train a deep learning model $\phi(x;\theta)$ that assigns large anomalous score to anomalies and small anomalous score to normal data, e.g., $\Omega(x)|_{y=-1} \geq \Omega(x)|_{y=1} + C$. To achieve this goal, we design a tailored hinge loss function that copes with anomalous and normal samples differently. More specifically, given the pseudo label $y$ of the training sample $x$, we define the loss function as:

$$l(x,y) = \begin{cases} \Omega(x), & y = 1 \\ \max\{C - \Omega(x), 0\}, & y = -1 \end{cases} \tag{4}$$

In Eq. 4, $\Omega(x)$ can be any anomalous score function used by existing unsupervised deep anomaly methods as discussed in Sec. 2.

Given the pseudo labels $y$ and the loss function defined in Eq. 4, ELITE learns the optimal parameters $\theta^*(y)$ to minimize the average loss incurred by these pseudo labels. The objective function is defined as follows.

$$\theta^*(y) = \arg\min_{\theta} \frac{1}{N^u} \sum_{i=1}^{N^u} l(x_i, y_i) \tag{5}$$

Given the pseudo labels $\hat{y}$, it is straightforward to learn $\theta^*(y)$, using the existing training methods.

Note in Eq. 4 $C$ is a hyper-parameters that controls the margin of anomalous score between normal samples and anomalies. The optimal parameters $\theta^*(y)$ will concurrently minimize the anomalous score of normal samples and grow the anomalous score of anomalies to a value no smaller than $C$.

An appropriate hyper-parameter $C$ is critical to the performance of ELITE. A too large $C$ tends to make the training process unstable, while a small $C$ fails to separate anomalies from normal samples. We design an intuitive method to automatically determine $C$. Given an unsupervised counterpart of ELITE denoted as $\phi(x;\theta^0)$ where $\theta^0$ represents its initial parameters, we simply set the hyper-parameter $C$ as its training loss averaged on all samples. The intuition is that because the training process targets minimizing the training loss on the normal examples, the final model will produce a training loss on each normal example that in average is guaranteed to be much smaller than the initial average loss. Therefore, a hyper-parameter $C$ set in this way tends to be effective in separating anomalies from normal samples.

### 3.2.2 Validation Loss

Given a set of labeled examples as validation set, ELITE defines the validation loss $\mathcal{L}^v$ as follows.

$$\mathcal{L}^v(\theta) = \frac{1}{N^l} \sum_{j=1}^{N^l} l(x_j^l, y_j^l; \theta) \tag{6}$$

In Eq. 6, $N^l$ represents the number of labeled examples and $l(x_j^l, y_j^l; \theta)$ corresponds to the training loss function (Eq. 4).

ELITE aims to assign a pseudo label $y$ to each unlabeled training sample so that the validation loss of the trained model is minimized.

$$y^* = \arg\min_y \mathcal{L}^v(\theta^*(y)) \tag{7}$$

Here $\theta^*(y)$ corresponds to the optimal parameters learned from the current pseudo labels as discussed in Sec. 3.2.1.

## 3.3 Pseudo Label Inference

The key of ELITE is to effectively identify the optimal pseudo labels that minimize the model's validation loss. Obviously, inferring such optimal pseudo labels by recursively flipping the label of each sample, re-training the deep anomaly model, and calculating the validation loss will be too expensive.

To solve this problem, ELITE proposes an efficient pseudo label inference method, called ALICE. The key idea is to use the gradient of the current model's validation loss to predict how altering the label of one training sample will change the validation loss.

### 3.3.1 Meta-gradient-based Pseudo Label Inference

Assume we have already trained a model using all training samples $\mathbb{X}^U$ and denote its learned parameters as $\theta^*$. Given a training sample $x_t$ in $\mathbb{X}^U$ , if we flip its label, we could learn a new model parameterized by $\theta^*_-$.

Let $L^v(\theta)$ denote the validation loss of a model parameterized by $\theta$, that is, the model's loss on the validation set. If we are aware of the difference between the validation loss of the original model $\theta^*$ and that of the new model $\theta^*_-$, namely, $L^v(\theta^*) - L^v(\theta^*_-)$, it will be straightforward to decide if we should flip the label of $x_t$. That is, assume $x_t$ was normal. If $L^v(\theta^*) - L^v(\theta^*_-) > 0$, ELITE should flip $x_t$ to be abnormal, and change its pseudo label as $\hat{y} = -1$, because this will reduce the validation loss. Otherwise, $x_t$ remains normal.

Because we already have $\theta^*$ of the original model, computing its validation loss $L^v(\theta^*)$ is straightforward, that is, $L^v(\theta^*) = \frac{1}{M} \sum_i^M l(x_i^l, y_i^l; \theta^*)$. The goal of ALICE is to estimate $L^v(\theta^*) - L^v(\theta^*_-)$ without learning the new model $\theta^*_-$.

By the objective function (Eq. 5), $\theta^*$ is learned as: $\arg\min_\theta L(\theta)$ where $L(\theta) = \frac{1}{N}[\sum_{i\neq t}^{N} l(x_i^u, y_i^u; \theta) + \Omega(x_t; \theta)]$. Here by the loss function (Eq. 4), $\Omega(x_t; \theta)$ represents the loss on $x_t$ if considering $x_t$ as normal.

Without loss of generality, we assume $C$ in the loss function (Eq. 4) is large enough and therefore $\max\{C - \Omega(x; \theta), 0\} = C - \Omega(x; \theta)$ that corresponds to the loss of an anomaly $x$. Now if we change $x_t$ to anomaly, the new model $\theta^*_-$ can be learned as follows:

$$\theta^*_- = \arg\min_\theta \{L(\theta) - \frac{2}{N}\Omega(x_t; \theta)\} \tag{8}$$

This is because altering the label of $x_t$ from normal to abnormal is equivalent to first removing $\Omega(x_t; \theta)$ from $L(\theta)$, and then adding $C - \Omega(x; \theta^*)$ back.

Next, we use $\epsilon$ to represent - $\frac{2}{N}$ that weights the training loss of $x_t$. Now Eq. 8 changes to: $\theta^*_- = \arg\min_\theta \{L(\theta) + \epsilon\, \Omega(x_t; \theta)\}$. Similar to [21, 54, 77], we consider $\epsilon$ as a variable [21]. Now $\theta^*_-$ is a function of $\epsilon$, denoted as $\theta(\epsilon)$. When $N$ is sufficiently large, $\epsilon$ approaches 0.

ALICE then uses the gradient of $\theta(\epsilon)$ at $\epsilon = 0$ to approximate the change from $L^v(\theta^*)$ to $L^v(\theta^*_-)$.

$$L^v(\theta^*) - L^v(\theta^*_-) = \frac{dL^v(\theta^*(\epsilon))}{d\epsilon}\Big|_{\epsilon=0} \tag{9}$$

We call the gradient $\mathcal{M} = \frac{dL^v(\theta^*(\epsilon))}{d\epsilon}\Big|_{\epsilon=0}$ as **meta-gradient**.

Once getting the meta-gradient, applying the update rule defined below is guaranteed to reduce the validation loss.

**Definition 3.1. Update Rule.**

$$\hat{y} = -\operatorname{sign}(L^v(\theta^*) - L^v(\theta^*_-)) = -\operatorname{sign}(\frac{dL^v(\theta^*(\epsilon))}{d\epsilon}\Big|_{\epsilon=0}) \tag{10}$$

The reason is that a positive value of $L^v(\theta^*_+) - L^v(\theta^*_-)$ means treating the new training sample as an anomaly will lead to a smaller validation loss than treating it as normal, and vice versa.

Note above we assume the training sample $x_t$ was originally normal. However, the update rule equally works if $x_t$ was abnormal.

### 3.3.2 Meta-gradient Estimation

To compute meta-gradient, the only thing missing here is $\theta^*(\epsilon)$. Similar to [77] ALICE approximates $\theta^*(\epsilon)$ by taking one step of gradient descent on the original model $\theta^*$.

$$\hat{\theta}(\epsilon) = \theta^* - \eta_\theta \epsilon \nabla_{\theta^*} \Omega(x_t, \theta^*) \tag{11}$$

$\eta_\theta$ represents leaning rate, a hyper-parameter of deep learning.

Given $\hat{\theta}(\epsilon)$, ALICE now is ready to apply the update rule to approximate $\hat{y}_i$. More specifically,

$$\begin{aligned}\hat{y}_i &= -\operatorname{sign}\left(\frac{dL^v(\hat{\theta}(\epsilon))}{d\epsilon}\Big|_{\epsilon=0}\right) \\ &= -\operatorname{sign}\left(\frac{d}{d\epsilon}\frac{1}{M}\sum_{i=1}^{M} l(x_i^l, y_i^l; \hat{\theta}(\epsilon))|_{\epsilon=0}\right)\end{aligned} \tag{12}$$

**Intuitive Interpretation of ALICE.** First, we unroll Equation 12 with the chain rule. Given a training sample $x_i$, we have $\hat{\theta}(\epsilon) = \theta^*$ when $\epsilon = 0$. Then we have:

$$\begin{aligned}\hat{y}_i &= -\operatorname{sign}\left(\frac{dL^v(\hat{\theta}(\epsilon))}{d\epsilon}\right) \\ &= \operatorname{sign}\left(\frac{L^v(\hat{\theta}(\epsilon))}{d\theta}\Big|_{\hat{\theta}(\epsilon)} \frac{d(\theta^* - \frac{1}{N}\eta_\theta \epsilon \nabla_\theta \Omega(x_i;\theta^*))}{d\epsilon}\Big|_{\epsilon}\right) \\ &= \operatorname{sign}\left(\frac{\eta_\theta}{N}\frac{dL^v(\theta^*)}{d\theta}\Big|_{\theta^*} \frac{d\Omega(x_i;\theta^*))}{d\theta}\Big|_{\theta^*}\right)\end{aligned} \tag{13}$$

Eq. 13 shows that $\hat{y}_i$ corresponds to an inner product between the gradient of the training loss of the given training sample and the gradient of the validation loss. Given a training sample $x_i$ initialized as normal, if its gradient is in the same direction to the gradient of the validation loss, then $x_i$ will indeed be a normal object. This is because in this case minimizing its training loss by gradient descent – the typical practice of deep learning optimization, will also minimize the validation loss. Otherwise, $x_i$ should be an anomaly.

### 3.3.3 Learning at Scale

**The Learning process.** Next, we introduce how ELITE infers the optimal pseudo labels for the entire unlabeled dataset. ELITE fuses ALICE into every iteration during the training process of the deep anomaly model and dynamically adjusts the labels of the training samples. ELITE starts with assuming that all unlabeled training samples are normal. Once one training iteration is done, ELITE estimates the meta-gradient for each sample $x_i$ and applies the update rule to update its pseudo label. Thereafter, ELITE updates the parameters of the deep anomaly model using Eq. 14.

$$\theta_{t+1} = \theta_t - \eta_\theta[\frac{1}{N}\sum_{i=1}^{N} \alpha_i \nabla_\theta l(x_i, \hat{y}_i; \theta)] \tag{14}$$

In Eq. 14, $\theta_{t+1}$ represents the new parameters, while $\theta_t$ represents the parameters produced in last iteration. $\eta_\theta$ is the learning rate. Same to the traditional gradient descent optimization, Eq. 14 uses the gradient of the loss function $\nabla_\theta l(x_i, \hat{y}_i; \theta)$ to update $\theta_t$. But

ELITE weights the meta-gradient at each training sample $x_i$ with $\alpha_i = \eta_M \cdot \|\mathcal{M}_i\|$, where $\|, \mathcal{M}_i\|$ represents the absolute value of the meta-gradient, and $\eta_M$ is a hyper-parameter. The intuition is that, if the meta-gradient of a training sample $x_i$ has a larger absolute value, $x_i$ is more important. This is because by our ALICE method, potentially $x_i$ will contribute more in reducing the validation loss.

**Batch Optimization.** Although ELITE effectively avoids recursively re-training the deep learning model, it still tends to be expensive when the unlabeled training dataset is large. Similar to [77, 41], ELITE employs a mini-batch based optimization strategy to address the efficiency concern. During each training iteration, ELITE randomly divides the unlabeled training samples into many mini-batches and then concurrently updates the labels with respect to each mini-batch. Each mini-batch contains only $n \ll N$ unlabeled objects. Therefore, it significantly speeds up the training process. As a standard deep learning training process, ELITE can run on any deep learning platform such as TensorFlow and Pytorch.

**Time Complexity Analysis.** Compared to unsupervised deep anomaly methods, ELITE requires an extra forward and backward pass to obtain the gradient of each training sample and an additional forward and backward pass to calculate $\hat{y}_i$. Thus, ELITE is approximately $3\times$ slower than the unsupervised deep anomaly methods. We argue that the additional computing cost is worthwhile in practice because ELITE is robust to polluted training data and significantly improves the accuracy of anomaly detection.

**Convergence Analysis.**

**Theorem 1.** *Suppose the validation loss $L^v(x;\theta)$ is Lipschitz smooth with constant $L$, and the gradient of training data is bounded by $\sigma$. Then as long as the learning rate $\eta_y\eta_\theta \leq \frac{2n}{L\sigma^2}$, the validation loss decreases monotonically,*

$$L^v(\theta_{t+1}) \leq L^v(\theta_t) \tag{15}$$

*Proof.* Without loss of generality, we assume $C$ in the loss function (Eq. 4) is large enough and therefore $\max\{C - \Omega(x;\theta), 0\} = C - \Omega(x;\theta)$ which corresponds to the loss of an anomaly $x$. Combining Equation 14 and Equation 13, we have,

$$\theta_{t+1} = \theta_t - \eta_y\eta_\theta\{\frac{1}{n}\sum_{i=i}^{n}[\nabla_\theta L^v(\theta_t)\nabla_\theta L_i(\theta_t)]\nabla_\theta L_i(\theta_t)\} \tag{16}$$

where $\nabla_\theta L^v(\theta_t) = \frac{\partial L^v(\theta_t)}{\partial\theta}\Big|_{\theta_t}$ and $\nabla_\theta L_i(\theta_t) = \frac{\partial\Omega(x_i;\theta_t))}{\partial\theta}\Big|_{\theta_t}$. For simplicity of expression, we denote $\nabla_\theta L^v(\theta_t)$ as $\nabla L^v$ and $\nabla_\theta L_i(\theta_t)$ as $\nabla L_i$.

Since the validation loss $L^v(\theta)$ is Lipschitz smooth with constant $L$, from [36],

$$L^v(\theta_{t+1}) \leq L^v(\theta_t) + (\nabla L^v)^T\triangle\theta + \frac{L}{2}\|\triangle\theta\|^2 \tag{17}$$

Plugging in Equation 16,

$$L^v(\theta_{t+1}) \leq L^v(\theta_t) - I_1 + I_2, \tag{18}$$

where,

$$I_1 = \eta_y \eta_\theta \sum_{i=1}^{m} (\nabla L^v \nabla L_i)^2 \tag{19}$$

and,

$$\begin{aligned} I_2 &= \frac{L}{2} \| \frac{\eta_y \eta_\theta}{n} \sum_{i=1}^{m} (\nabla L^v \nabla L_i) \nabla L_i \|^2 \\ &\leq \frac{L}{2} \frac{\eta_y^2 \eta_\theta^2}{n^2} \sum_{i=1}^{m} \| (\nabla L^v \nabla L_i) \nabla L_i \|^2 \\ &= \frac{L}{2} \frac{\eta_y^2 \eta_\theta^2}{n^2} \sum_{i=1}^{m} (\nabla L^v \nabla L_i)^2 \| \nabla L_i \|^2 \\ &\leq \frac{L}{2} \frac{\eta_y^2 \eta_\theta^2}{n^2} \sum_{i=1}^{m} (\nabla L^v \nabla L_i)^2 \sigma^2 \end{aligned} \tag{20}$$

The first inequality comes from the triangle inequality, and the second inequality holds since the gradient of training data is bounded by $\sigma$. If we denote a value $\tau$ at iteration $t$, $\tau_t = \sum_{i=1}^{m} (\nabla L^v \nabla L_i)^2$, then we have,

$$L^v(\theta_{t+1}) \leq L^v(\theta_t) - \frac{\eta_y \eta_\theta}{n} \tau_t (1 - \frac{L \eta_y \eta_\theta \sigma^2}{2n}) \tag{21}$$

Note by definition $\tau_t$ is non-negative and $\eta_y \eta_\theta \leq \frac{2n}{L\sigma^2}$, we have,

$$L^v(\theta_{t+1}) \leq L^v(\theta_t) \tag{22}$$

Theorem 1 is proven. □

## 3.4 Example: Applying ELITE to Deep SVDD

In this section, we show that ELITE is able to easily adapt existing unsupervised deep anomaly methods to benefit from the anomaly examples at hand. More specially, to support one unsupervised deep anomaly method, the only change we need to make is to plug its anomalous score function $\omega(x)$ into the loss function of ELITE (Eq. 4 in Sec. 3.2). Next, we use Deep SVDD [81] as an example to showcase this. Deep SVDD is briefly reviewed in Sec. 2.

As shown in Algorithm 1, ELITE starts with initializing the neural network's parameters $\theta$ and the hypersphere center $o$ exactly as what Deep SVDD does. Then, in each epoch ELITE samples a mini-batch of unlabeled samples $B_U$ and uses the labeled samples as validation set. Next, ELITE assigns an initial pseudo label $\hat{y}_i$ to each unlabeled

**Algorithm 1** ELITE on Deep SVDD

**Input:**
Unlabeled data: $X_U : \{x_1, \ldots, x_N\}$
Validation examples: $X_V : (\{(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_M, \tilde{y}_M)\}$
Hyperparameters: $\eta_M, \eta_\theta$, Hypersphere center, $o$, Margin, $C$
Loss Function: $\Omega(x, o) = \|x - o\|$

**Initialize:**
  Neural network weights: $\theta$
**for** each epoch **do**
  **for** each mini-batch **do**
    Draw mini-batch $B_U$:$\{x_1, \ldots, x_n\}$ from $X_U$
    Draw mini-batch $B_V$:$\{(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_m, \tilde{y}_m)\}$ from $X_V$
    **Initialize:**
    $\hat{y}_i \leftarrow 0 \; \forall x_i \in B_U$
    $\hat{\theta}(\hat{y}) \leftarrow \theta - \eta_\theta[\frac{1}{n} \sum_{i=0}^{n} \hat{y}_i \nabla_\theta \Omega(x, \theta)]$
    **Update:**
    $\mathcal{M}_i \leftarrow \eta_y \frac{\partial}{\partial y_i} \frac{1}{m} \sum_{i=1}^{m} L^v(\tilde{x}_i, \hat{\theta}(\hat{y}))|_{\tilde{y}})$
    $\hat{y}_i = -\operatorname{sign}(\mathcal{M}_i)$
    $\alpha_i = \eta_M \cdot \|\mathcal{M}_i\|$
    $\theta \leftarrow \theta - \eta_\theta[\frac{1}{n} \sum_{i=0}^{n} \alpha_i \nabla_\theta l(x_i, \hat{y}_i; \theta)]$

**Output:** Trained Model: $\phi^\star(x, \theta^\star)$

sample in $B_U$. ELITE uses these pseudo labels to learn the parameters $\theta$ of the network. It then computes the validation loss using the loss function in Eq. 4, alters the pseudo labels according to the update rule in Def. 3.1, and updates the parameters by Eq. 14. These steps iterate until the validation loss is minimized or reaching the epoch limit.

# 4 Related Works

**Unsupervised Deep Anomaly Detection.** Unsupervised deep anomaly techniques in general can be characterized into two categories. The first category learns a representation that better distinguishes anomalies from normal data. Some of these techniques [42, 84, 7, 122, 18] use the reconstruction errors of Auto-Encoder as the anomalous score to directly detect anomalies, assuming that Auto-Encoders incur larger reconstruction errors on anomalies than normal objects. Some other techniques use the same principle, but apply different deep learning techniques to learn the data representation, such as Generative Adversarial Networks [73, 121, 4], self-learning models [37] and Auto-regressive models [2]. One-class classification-based methods [86, 81, 82, 29, 83] instead learn a feature embedding that maps normal objects into a minimal volume hyper-sphere; then the objects out of the hyper-sphere are considered as anomalies. The second category of deep anomaly techniques [97, 90, 129, 102] use learned deep embedding to enhance the classical shallow anomaly detection methods. To learn a representation that is effective in separating anomalies, most of these methods require a clean training data set – a data set not containing any anomalies. However, such clean training data rarely exist in real applications.

**Robust Deep Anomaly Detection.** Robust deep anomaly detection [128, 10, 113, 16] targets this problem. Based on the assumption that anomalies in the training samples tend to incur large training loss in the training process, these techniques iteratively remove anomalies from the training set in each training epoch. However, they suffer from the chicken-egg problem. That is, identifying anomalies based on the training loss requires an accurate model, while training an accurate model needs a clean training set. Another strategy is to use the deep learning techniques that are robust to anomalies [57, 29] to learn the representation. However, to overcome the influence of anomalies these techniques often assume the distribution of the normal examples is known beforehand. This assumption usually does not hold in practice.

**Semi-supervised Deep Anomaly Detection** Semi-supervised deep anomaly detection [82, 46, 40] uses a small number of anomaly examples to improve the accuracy of unsupervised deep anomaly techniques. Similar to classical semi-supervised classification, their key idea is to use these anomaly examples as *labeled training data* that are modeled as *labeled loss* to supplement the loss function of the unsupervised deep learning method. However, these techniques still assume that the unlabeled training data is clean and essentially treat them as labeled normal examples. Therefore, they suffer from the performance degradation caused by the hidden anomalies in the unlabeled training data. Our ELITE approach instead uses a small set of anomaly examples as *validation set*. It effectively discovers the anomalies hidden in the polluted training data and turns these anomalies into useful signals that help to learn a data representation that better distinguishes between normal and abnormal samples.

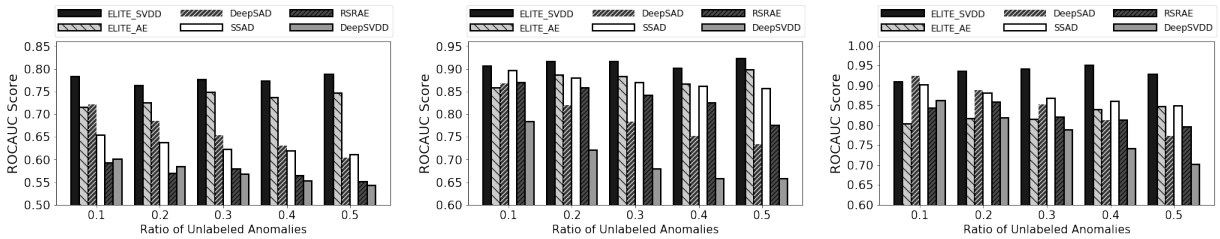
# 5 Experiments

We conduct an experimental study to evaluate the effectiveness of ELITE. Specifically, we focus on the following four questions:

1. **Robustness to Polluted Training Data**: How does ELITE compare with existing deep anomaly techniques in term of the robustness to the polluted training data?
2. **Performance with different number of labels**: How does ELITE perform in contrast to the existing deep anomaly methods when using different number of labels?
3. **Sensitivity Analysis**: Is ELITE sensitive to the selection of its hyper-parameters?
4. **Training Mechanism**: How is our training mechanism different from the standard semi-supervised learning?

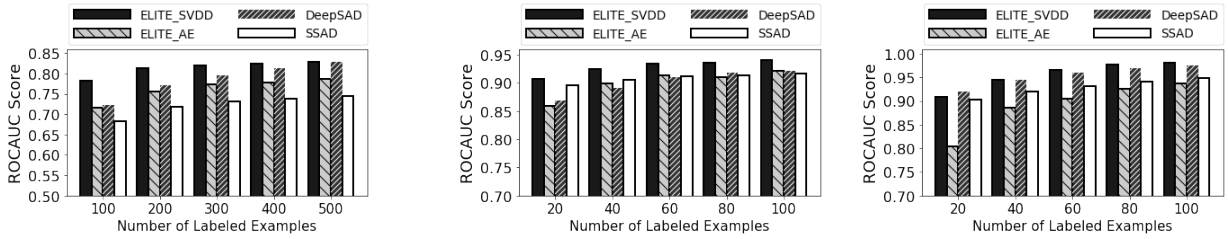
## 5.1 Experiment Setup and Methodology

**Experimental Setup.** All experiments are conducted on Google Cloud with a virtual machine with 12 CPU cores and 4 P-100 GPUs. All code is developed with Python 3 on Pytorch 1.5.0.




(a) **CIFAR-10** (b) **FMNIST** (c) **MNIST**

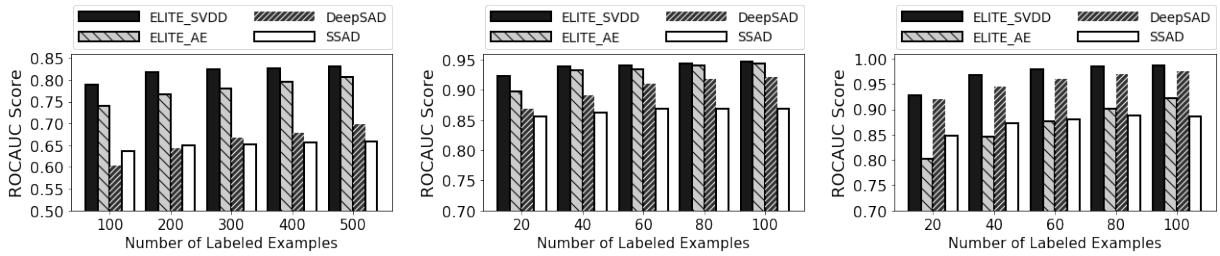
Figure 4: ROCAUC: Varying the Ratio of Anomalies in Training data




(a) **CIFAR-10 (**$r_p = 0.1$**)** (b) **FMNIST (**$r_p = 0.1$**)** (c) **MNIST** ($r_p = 0.1$)

Figure 5: ROCAUC: Varying the Number of Labeled Examples (Lightly Polluted)

**Datasets.** We evaluate ELITE using three benchmark datasets which are also frequently used in the experiments of the state-of-the-art deep anomaly works we compare against [82, 81].




(a) **CIFAR-10**($r_p = 0.5$) (b) **FMNIST (**$r_p = 0.5$**)** (c) **MNIST (**$r_p = 0.5$**)**

Figure 6: ROCAUC: Varying the Number of Labeled Examples (Heavily Polluted)

• **MNIST:** The MNIST dataset consists of 28 $\times$ 28 pixel grayscale images of the handwritten digits 0-9. Each image contains only one digit centered in the frame and is given a class label corresponding to the digit it contains. Given the relatively simple and clear shape of the digits and the consistent black background, we consider it as the least complex dataset among the three datasets we use.
• **FMNIST:** The FMNIST or Fashion-MNIST dataset was created to be a more complex replacement for MNIST. FMNIST consists of 28x28 pixel grayscale images for ten types of clothing articles such as T-shirts, coats, and sneakers with corresponding labels.
• **CIFAR-10:** The CIFAR-10 dataset consists of 32x32 color images of ten distinct object classes. Four of the classes are types of vehicles – airplane, automobile, ship, truck – with the remaining six being varying types of animals. Images in this dataset were originally drawn from internet search engines and converted to the 32x32 resolution.

**Alternative Methods.** We compare ELITE against the state-of-the-art unsupervised (DeepSVDD [81]), semi-supervised (DeepSAD [82], SSAD [40], and robust (RSRAE [57]) deep anomaly methods. Moreover, to show ELITE is model agnostic, we implement ELITE on top of two types of unsupervised deep anomaly models, namely the one-class classification-based DeepSVDD [81] and Auto-Encoder.
• **DeepSVDD** [81] is the state-of-the-art unsupervised anomaly method, which detects anomalies by mapping the training data into a compact hyper-sphere, assuming the training data is clean.
• **DeepSAD** [82] extends Deep SVDD method to the semi-supervised setting and uses the labeled examples as training data to improve the accuracy of anomaly detection. We consider DeepSAD as the most related work to ELITE.
• **SSAD** [40] is a popular shallow semi-supervised anomaly method built on vanilla SVDD [100]. Similar to DeepSAD, it directly uses the labeled examples as training data and encourages the model to generate large anomalous score on the labeled anomalies.
• **RSRAE** is the state-of-the-art robust deep anomaly method, which combines a simple Auto-Encoder with robust deep learning techniques, more specifically Robust Subspace Recovery (RSR) layer. The RSR layer is used to learn a subspace within the latent space where normal and anomalous samples are well separated.

**Methodology.** Following the state-of-the-art [82], for each dataset we select one class as normal and consider other classes as abnormal. To ensure that results are not class dependent, we repeat each set of experiments with a different class selected as the normal class until all classes are exhausted. We then report the average of these results. For each experiment, we randomly select 5,000 objects to create a training dataset. This set contains samples from both normal and anomalous classes, with the ratio of anomalies controlled by the value $r_p$. In general $r_p$ is selected to be small such that the majority of the training samples are drawn from the normal class, while the few anomalies are drawn from the remaining classes. From each dataset, we randomly sample an equal number of normal samples and anomalies to be used as the labeled training dataset and consider the remaining samples to be unlabeled. We vary the ratio of training points allocated to the labeled training set $r_l$ and the ratio of pollution in the training dataset $r_p$ to analyze the performance of ELITE in a wide variety of scenarios. Again, following [82], we use the Area Under Curve (AUC) score of the Receiver Operating Characteristic (ROC) curve as the metric to evaluate the accuracy of each method.

## 5.2 Varying the Ratio of Anomalies

In this experiment, we investigate the robustness of different deep anomaly detection methods to the increasing ratio of anomalies in the training set. To do this, we vary the ratio of anomalies in training set from 0.1 to 0.5. We fix the ratio of labeled examples $r_l$, and repeat the experiments on all ten classes and report the average results over all experiments on each dataset. For MNIST and FMNIST we use 20 labeled examples, while for CIFAR-10 we use 100 to account for its much higher complexity.

Figure 4 indicates that both of our ELITE-based methods, ELITE _AE and ELITE _SVDD, outperform all other methods by up to 30%, especially on the complex datasets such as CIFAR-10. Also, we find that the performance of ELITE never degrades with the increasing ratio of anomalies in training data. However, the performance of the state-of-the-art methods, including the robust deep anomaly method RSRAE, significantly decrease as the ratio of anomalies in the training data increases. Furthermore, on the *CIFAR-10* and *FMNIST* dataset, ELITE achieves even higher performance when the anomaly ratio is highest, i.e. $r_p = 0.5$. This is because ELITE not only identifies the anomalies in the training dataset, but also effectively uses them to learn an anomaly-aware data representation that improves the accuracy of anomaly detection. This confirms that ELITE not only outperforms the other methods but also is much more robust to anomalies in the training dataset. Furthermore, we find that the shallow *SSAD* method even outperforms its deep competitor, *DeepSAD*. We argue that this shows it is easier for deep anomaly detection models to overfit the anomalies in the training data due to their complex network structure using a large number of parameters.

## 5.3 Varying the Ratio of Labeled Examples

In this scenario, we compare the performance of different semi-supervised deep anomaly methods given a different number of labeled examples. For this experiment, we evaluate our method on both lightly polluted training data where $r_p = 0.1$, and heavily polluted training data where $r_p = 0.5$. For FMNIST and MNIST we vary the number of labeled samples from 20 to 100 in steps of 10 ($r_l = 0.004 - 0.02$), while for CIFAR-10 we test 100 to 500 labeled samples with intervals of 50 ($r_l = 0.02 - 0.1$). Again, we exhaustively use every class in each dataset as normal samples and report each dataset's average result.

Figure 5 and Figure 6 show the result on lightly polluted ($r_p = 0.1$) and heavily polluted datasets ($r_p = 0.5$) respectively. Both of our methods significantly outperform the other methods on all heavily polluted datasets by up to 25%. This again shows ELITE is significantly more robust to anomalies in the training data, because ELITE effectively leverages the labeled examples. Moreover, ELITE reaches very high accuracy with very few labeled examples. This is because ELITE uses the labeled examples as validation data, and it requires much fewer labels to evaluate the model performance than training the model. Therefore, although increasing the number of labels improves the performance of *DeepSAD*, it is consistently less accurate than our ELITE-based methods. Note that even when the dataset is lightly polluted, *DeepSAD* still requires 2 - 3 times more labeled examples to achieve comparable performance to ELITE on complex datasets like *CIFAR-10* and *Fashion-MNIST*.

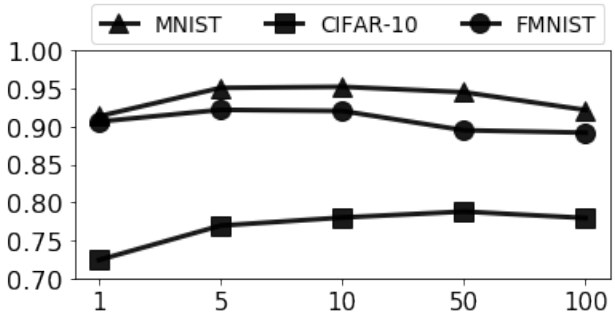
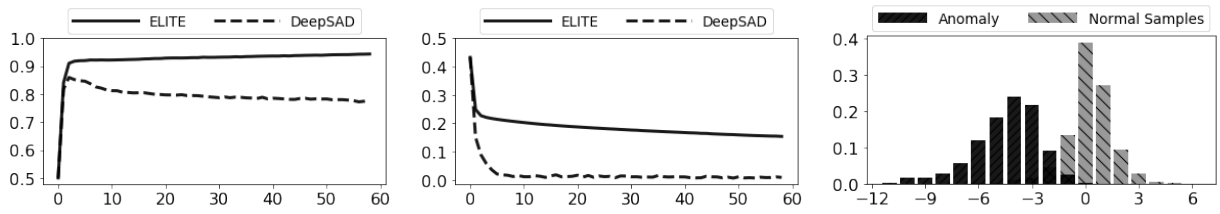



Figure 7: **Sensitivity Analysis of $\eta_M$ of ELITE**




(a) **ROCAUC VS Training Epochs** (b) **Labeled Loss VS Training Epochs** (c) **Distribution of Anomalous Scores**

Figure 8: ROCAUC: Varying the Number of Labeled Examples

## 5.4 Sensitivity Analysis

Here we investigate how sensitive ELITE is to the value of hyper-parameter $\eta_M$ which controls the factor that the validation loss plays in the learning process. We report the results on our ELITE _SVDD method, although ELITE _AE shows the similar trend. We set $r_p$ to 0.1 and we use 20 labeled examples for both MNIST and FMNIST and 100 labeled examples for CIFAR-10. We vary $\eta_M$ from 1 to 100, while keeping all other hyper-parameters fixed. Figure 7 show that the performance of ELITE is stable. This confirms that ELITE is not sensitive to the hyper-parameter $\eta_y$, and thus partially mitigates the hyper-parameter tuning problem. We also observe that *FMNIST* and *MNIST* prefer small $\eta_M$ as the performance decreases with the increase of $\eta_M$. However, on CIFAR-10 ELITE achieves slightly better performance as $\eta_M$ increases. Therefore, based on these results, we recommend to set a large $\eta_M$ on complex datasets and set a small value if the data set is relatively simple.

## 5.5 Evaluating the Training Mechanism

### 5.5.1 Training Process

To better understand the training mechanism of ELITE, we compare ELITE with the semi-supervised *DeepSAD* which is based on the classical semi-supervised classification mechanism. To ensure a fair comparison, we apply the same loss function (Eq. 4) to both ELITE and *DeepSAD*. We report the results on the *FMNIST* dataset. Figure 8(a) and Figure 8(b) depict how ROCAUC score and labeled loss change over the training process. In *DeepSAD*, the loss on labeled examples quickly decreases to 0, while it reduces slowly in ELITE. Meanwhile, the ROCAUC score of *DeepSAD* decreases after reaching the peak, potentially because the deep neural network starts overfitting the labeled examples. In contrast, the ROCAUC score of ELITE increases stably.

### 5.5.2 Distribution of Anomalous Scores

As discussed in Sec. 3.3, ALICE, ELITE's label inference method, uses meta-gradient to determine the anomalous score of the training data, because the meta-gradient of anomalies tends to show distinct patterns from that of normal samples. Here we verify its effectiveness by measuring the distribution of $\hat{y} \cdot \|\mathcal{M}\|$ which represents the anomalous score of each training sample. In this experiment, we run ELITE on *MNIST* with $r_p = 0.5$ and $r_l = 0.004$. We separately report the $\hat{y} \cdot \|\mathcal{M}\|$ of normal and anomalous samples averaged over the first 500 iterations. Figure 8(c) shows that the anomalous score effectively separates anomalous samples from normal ones. That is, ELITE assigns small scores (negative) to anomalous samples, while large scores (positive) to normal samples. Although ELITE still erroneously assigns negative score to some normal samples, their scores still tend to be larger than those of the real anomalies. This confirms the effectiveness of our ALICE method.

# Part II
# LANCET: Labeling Complex Data At Scale

(The system described in this part has been accepted at VLDB2021.)

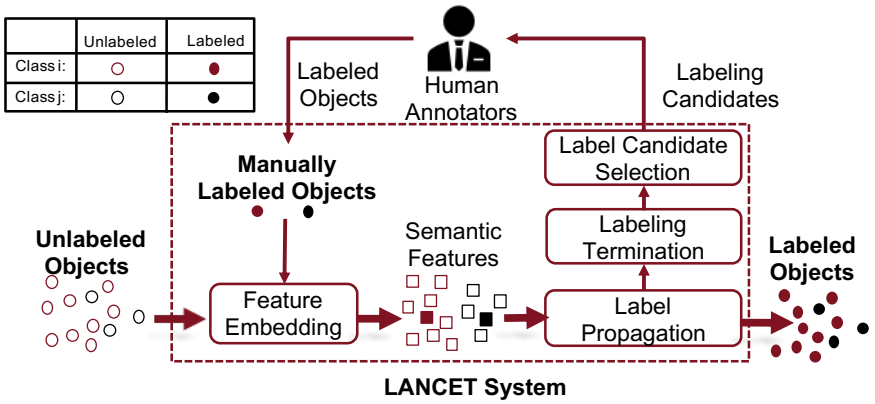



Figure 9: Overall Process of LANCET

## 6 Overall Process of LANCET

LANCET targets the multi-class classification problem defined over an input space $\mathcal{X} \in \mathrm{R}$ and a label space $\mathcal{Y} = \{1, ..., C\}$. The objective of LANCET is to use minimal human labeling efforts to produce labels that are sufficient to solve the classification problem.

As shown in Fig. 9, LANCET consists of four components, namely feature embedding, label propagation, label candidate selection, and labeling termination. Next, we briefly introduce how to use LANCET to solve the labeling problem.

Labeling in LANCET is an iterative process. Using the existing manually labeled objects, LANCET first employs the feature embedding component to project the raw input data into a semantic feature space that takes the classification task into consideration. On top of the semantic feature space, LANCET uses the label propagation component to propagate labels from the manually labeled objects to the unlabeled objects. The labeling termination component then determines if existing labels are already sufficient to train an accurate classification model. If not, LANCET uses the label candidate selection component to select at most $b$ objects as candidates for manual labeling, where $b$ is a user-controlled parameter.

In the next iteration, LANCET uses the enriched pool of manually labeled objects to update the semantic feature space and continues the labeling process. It stops when the

labeling termination component determines that continuing to label will not lead to clear performance gains.

# 7 LANCET Theoretical Foundation

In this section, we establish the theoretical foundation for LANCET. The key insight is that as long as the distributions of both the labeled objects and unlabeled objects satisfy certain statistical properties, we can accurately infer the labels of the unlabeled objects. This finding guides the design of our strategies for feature embedding, label propagation, and label candidate selection.

We denote the labeled objects as $\mathcal{D}_l = \{(x_l^i, y_l^i)\}_{i=1}^{N_l}$ and unlabeled objects as $\mathcal{D}_u = \{x_u^i\}_{i=1}^{N_u}$, where $y_l^i$ is the label of object $x_l^i$, and $N_l$ and $N_u$ denote the number of labeled and unlabeled objects, respectively. We assume the labeled objects are sampled from the joint probability distribution $P_l(x, y)$ in domain $\mathcal{X} \times \mathcal{Y}$. Accordingly, the unlabeled data objects are sampled from the joint probability distribution $P_u(x, y)$. Note that in practice, we do not have access to their ground truth labels for the unlabeled data objects $\{y_u^i\}_{i=1}^{N_u}$.

We show that, if the *labeled* and *unlabeled* objects jointly satisfy certain conditions detailed below, we can accurately predict the label $\hat{y}_u$ of each unlabeled object $x_u$. More formally, as long as these conditions hold, there exists a classification model $\phi(x)$ that makes Eq. 23 hold.

$$\mathbf{E}_{(x,y)\in P_u(x,y)}[loss(\hat{y}, y)] = \int_x p_u(x) loss(\hat{y}, y) dx < \epsilon \tag{23}$$

In Eq. 23, $\epsilon$ is a positive value that can be arbitrarily small. $\hat{y}$ represents the label w.r.t. each object $x$ predicted by $\phi(x)$. $loss(\hat{y}, y)$ corresponds to the cross entropy loss between the prediction $\hat{y} = \phi(x)$ and the ground truth label $y$, as defined in Eq. 24.

$$loss(\hat{y}, y) = -\sum_{i=1}^{C} p_u(y_i|x) log p(\hat{y}_i|x) \tag{24}$$

In other words, the two conditions introduced below ensure that the label propagated to the unlabeled objects has an *expected* cross entropy loss *close to 0* and hence is near perfect. Next, we introduce these two conditions and then formally prove the above *sufficiency* claim in Lemma 2.

**Definition 7.1. The Covariate-shift condition** [119, 91], expressed by Eq. 25, indicates that the joint distribution of unlabeled objects $p_u(x, y)$ and that of labeled objects $p_l(x, y)$ should have the same conditional probability mass function given a value $x$ in the feature space $\mathcal{X}$.

$$p_u(y|x) = p_l(y|x) \tag{25}$$

Intuitively, if the Covariate-shift condition holds, highly likely an unlabeled object will share the label with its close labeled neighbors. Otherwise, it would be impossible to accurately infer the labels of unlabeled objects no matter how many labeled objects were to exist. This is because machine learning models tend to infer the class of an unlabeled object based on its similarity to labeled objects. This Covariate-shift condition guides us

to develop the *feature embedding* approach to simplify the label generation problem, as described in Sec. 8.

**Continuity Condition.** The Continuity condition is based on the concept of Radon-Nikodym derivative (RND) [119].

**Definition 7.2.** The Radon-Nikodym derivative (RND) [119] is denoted as $\beta(x) = \frac{p_u(x)}{p_l(x)}$, where $p_u(x)$ and $p_l(x)$ represent the probability density functions (PDF) of unlabeled objects and labeled objects respectively. The **Continuity condition** holds if $\beta(x) < B$ and $B \ll \infty$.

The Radon-Nikodym derivative (RND) is also called the *importance weight* or *density ratio* in the literature [119, 63]. RND is not well defined if there exists an unlabeled object with coordinate $x$ such that $x$ has a large value on the PDF of the unlabeled objects ($p_u(x) > 0$), but a very small value on the PDF of the labeled objects ($p_l(x) \approx 0$). Intuitive, in this case this object does not have close labeled neighbors to represent it. Driven by this Continuity condition, we design a *label candidate selection* method (Sec. 10).

Based on the definitions of Covariate-shift condition and Continuity condition, we are ready to prove the key sufficiency claim.

**Lemma 2.** Assume in a feature space $\mathcal{X}$, the unlabeled objects $x_u \in \mathcal{D}_u$ and labeled objects $x_l \in \mathcal{D}_l$ satisfy the Covariate-shift and Continuity conditions. Then given a classification model $\phi(x)$ which infers the label of each object $\hat{y}$, if $\int_{x \in \mathcal{D}_l} p_l(x) loss(\hat{y_l}, y_l))dx \leq \frac{\epsilon}{B}$,then $\int_{x \in \mathcal{D}_u} p_u(x) loss(\hat{y_u}, y_u))dx \leq \epsilon$, where $\epsilon$ corresponds to a positive value that can be arbitrarily small and B is the threshold used to define Continuity condition.

*Proof.* When the Covariate-shift and Continuity conditions hold, the expected cross-entropy loss on the unlabeled objects can be easily estimated based on the training loss of the labeled objects and the RND value $\beta(x)$ used in Continuity condition.

$$\begin{aligned}
\int_x p_u(x) loss(\hat{y_u}, y_u)dx &= -\int_x p_u(x)[\sum_{i=1}^{C} p_u(y_i|x) logp(\hat{y}_i|x)]dx \\
&= -\int_x \frac{p_u(x)}{p_l(x)} p_l(x)[\sum_{i=1}^{C} p_l(y_i|x) logp(\hat{y}_i|x)]dx \\
&= \int_x \beta(x) p_l(x) loss(\hat{y_l}, y_l)dx \\
&\leq B \int_x p_l(x) loss(\hat{y_l}, y_l)dx \leq \epsilon
\end{aligned} \tag{26}$$

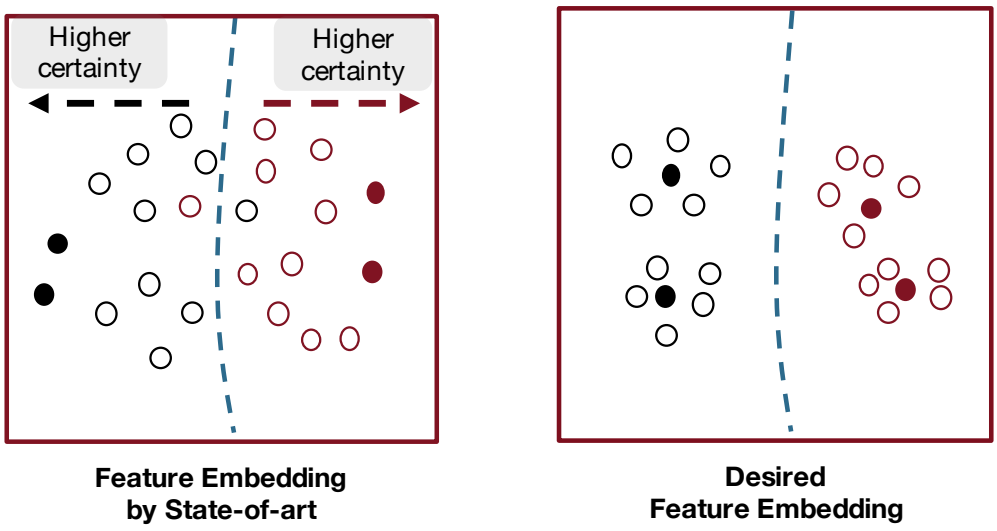
Lemma 2 is proven. □

Lemma 2 shows that a model $\phi(x)$ will be effective at propagating labels in a feature space that satisfy the Covariate-shift and Continuity conditions, if $\phi(x)$ is able to minimize the training loss on the labeled objects close to 0. Such a model $\phi(x)$ tends to exist

in practice. This is because it is widely observed [123] that machine learning models, especially deep learning models, can perfectly fit any training dataset even if its labels are randomly produced. In other words, they are able to minimize the training loss to 0.

However, usually the Covariate-shift and Continuity conditions do not naturally hold in the real world, complex data sets. The goal of LANCET is thus to make the data satisfy these conditions and ensure the effectiveness of the downstream classification tasks by transforming the data space (Sec. 8), effectively propagating labels (Sec. 9), and smartly selecting objects for human to labels (Sec. 10).

# 8 Feature Embedding




(a) (b)

Figure 10: (a) The semantic features extracted by existing semi-supervised models [85, 22]. Because theses models tend to overfit the labeled objects by pushing them far from decision boundaries, the semantic features of labeled and unlabeled data diverge from each other. (b) The desired semantic features to perform label propagation. The labeled and unlabeled objects should be close to each other if they fall into the same class.

According to the Covariate-shift condition, to accurately infer the labels of the unlabeled objects, any unlabeled object should share the label of its close labeled neighbors as shown in Fig. 10 (b). However, this often does not hold in the raw feature space of the input data, especially for the highly complex, high dimensional data such as images or time series. This is because the raw features of the complex data typically are not informative at distinguishing between the objects belonging to different classes [39]. Thus, the objects are not naturally separable by their classes.

To solve this problem, we develop a feature embedding method customized for our labeling task, called conditional feature matching or CFM for short. Guided by the small number of labels at hand, CFM projects the input data into a semantic feature space $\mathcal{Z}$ in which Eq. 25 holds, as required by the Covariate-shift condition.

## 8.1 Conditional Feature Matching

Our goal is to learn a feature embedding model $\mathcal{M}(x)$ that maps the input data $\mathcal{X}$ into a semantic feature space $\mathcal{Z}$ satisfying the Covariate-shift condition. LANCET learns $\mathcal{M}(x)$ based on both the labeled objects $\mathcal{D}_l$ and unlabeled objects $\mathcal{D}_u$. We use both classes of

objects for two reasons. First, the Covariate-shift condition reflects the statistical properties of both labeled and unlabeled objects. Therefore, the feature embedding model has to take the labeled objects into consideration. Purely unsupervised feature embedding method such as AutoEncoders [106, 96] do not satisfy this need. Second, in the labeling task, not many labeled objects are available apriori. Therefore, it is not practical to train a purely supervised feature embedding model.

Intuitively, semi-supervised classification techniques such as SemiGAN [85, 22] appear to be a natural solution to this problem. Semi-supervised classification learns a classification model by jointly minimizing the loss incurred by both the labeled objects and unlabeled objects, namely the labeled loss and the unlabeled loss. Similar to the classical deep learning models, the learned classification model $\mathcal{F}(x, \theta)$ automates the feature extraction process. Therefore, logically it can be decomposed into a feature extractor $\mathcal{M}$ and a classifier $f$. That is, $\mathcal{F}(x, \theta) = f(\mathcal{M}(x))$. $\mathcal{M}(x)$ produces a semantic feature space $\mathcal{Z}$.

**The Insufficiency of Existing Semi-supervised Feature Embedding.** Although the semantic features learned in this way is shown to be effective at separating objects belonging to different classes, they do not necessarily respect the Covariate-shift condition. In particular, we observe that these techniques tend to overly minimize the labeled loss by pushing the labeled objects far from the classification decision boundary in the learned semantic feature space. As a consequence, the labeled objects often are isolated far from the unlabeled objects, thus violating the Covariate-shift condition, as depicted in Fig. 10 (a).

**Solution: Condition Feature Matching.** To solve this problem, we propose the conditional feature matching strategy. The key idea is to express the Covariate-shift condition defined in Eq. 25 as a conditional feature matching (CFM) loss (Eq. 27). This way, it can be seamlessly plugged into the loss function of the semi-classification model $\mathcal{F}(x, \theta)$ which originally had only included the labeled loss and the unlabeled loss. As a regularization of the learning process, this conditional feature matching loss enforces that the unlabeled objects are close to the labeled objects in the semantic feature space $\mathcal{Z}$ if they belong to the same class; and it satisfies the Covariate-shift condition with a theoretical guarantee.

In the semantic feature space $\mathcal{Z}$, we denote a labeled object as $z^i_{l,c_m}$ if it belongs to class $c_m$. We use $N_{l,c_m}$ to represent the number of labeled objects that belong to class $c_m$. Accordingly, we denote one unlabeled object as $z^i_{u,c_m}$ if it is classified to class $c_m$. $N_{u,c_m}$ denotes the number of unlabeled objects classified to class $c_m$. Next, we formally define the CFM loss.

$$loss_{cfm} = \sum_{c_m=1}^{\mathcal{C}} [\|\mu_{l,c_m} - \mu_{u,c_m}\| + \sum_{n \neq m}^{\mathcal{C}} \max\{0, \lambda - \|\mu_{l,c_m} - \mu_{u,c_n}\|\}] \tag{27}$$

In Eq. 27, $\mu_{l,c_m} = \frac{1}{N_{l,c_m}} \sum_{i=1}^{N_{l,c_m}} z^i_{l,c_m}$ represents the center of all class $c_m$ *labeled* objects, while $\mu_{u,c_m} = \frac{1}{N_{u,c_m}} \sum_{i=1}^{N_{u,c_m}} z^i_{u,c_m}$ represents the center of all *unlabeled* objects that are predicted as class $c_m$. $\lambda$ controls the distance between $\mu_{u,c_m}$ and $\mu_{u,c_n}$, where $c_m$ and $c_n$ denote two different classes. Minimizing Eq. 27 thus will encourage the unlabeled and the

labeled objects to centralize into the same region in the semantic feature space if they potentially share the same label, and push them far away from each other if they belong to different classes.

**Theoretical Guarantee.** We first show that an unlabeled object is guaranteed to share the label with its close labeled neighbor if we can minimize the CFM loss to 0. Then we establish the connection between the CFM loss and the Covariate-shift condition.

Assume the unlabeled objects in class $c_m$ follow the Gaussian distribution. So do the labeled objects. The distributions are independent to each other. Their covariance matrices are the unit matrix, i.e. $\{z^i_{l,c_m}\}_{i=0}^{N_{l,c_m}} \sim \mathcal{N}(\mu_{l,c_m}, I)$, $\{z^j_{u,c_m}\}_{j=0}^{N_{u,c_m}} \sim \mathcal{N}(\mu_{u,c_m}, I)$. Given a pair of labeled and unlabeled objects $(z^i_l, z^j_u)$, we use $c(i=j)$ to represent that they belong to the same class and use $P^+_{ij}$ to denote its probability. Accordingly we use $c(i \neq j)$ to represent the pair of objects belonging to different class and denote its probability as $P^-_{ij}$. We use $d_{ij}$ to denote $z^i_l - z^j_u$.

**Lemma 3.** Assume the labeled objects $\{z^i_{l,c_m}\}_{i=0}^{N_{l,c_m}}$ and unlabeled objects $\{z^j_{u,c_m}\}_{j=0}^{N_{u,c_m}}$ of class $c_m$ independently show Gaussian distribution Then given a pair of labeled and unlabeled objects $(z^i_l, z^j_u)$ where $\|d_{ij}\| < \epsilon$ with $\epsilon$ denoting a small value close to 0, if the CFM loss is minimized to 0, then $P(c(i=j)\big|\|d_{ij}\| < \epsilon) \geq \frac{P^+_{ij}}{P^+_{ij} + \sum_{i \neq j} P^-_{ij} e^{-\frac{\lambda^2}{4}}}$.

*Proof.* By [61], $d_{ij}$ follows the Gaussian Distribution $d_{ij} \sim \mathcal{N}(\mu_{l,c_m} - \mu_{u,c_n}, 2I)$, where $z^i_l$ belongs to class $c_m$ and $z^j_u$ belongs to class $c_n$. Then the probability that a pair of labeled and unlabeled objects $(z^i_l, z^j_u)$ has a distance smaller than $\epsilon$ is,

$$P(\|d_{ij}\| < \epsilon) = \frac{1}{\sqrt{4\pi}} \int_{\epsilon} e^{-\frac{1}{4}[x - (\mu_{l,c_m} - \mu_{u,c_n})]^2} dx = \frac{1}{\sqrt{4\pi}} e^{-\frac{1}{4}\|\mu_{l,c_m} - \mu_{u,c_n}\|^2} \tag{28}$$

Because the CFM loss is minimized to 0, $\forall\ c_m$, $\mu_{l,c_m} = \mu_{u,c_m}$, and $\forall\ c_m \neq c_n$, $\|\mu_{l,c_m} - \mu_{u,c_n}\| \geq \lambda$.

Given a labeled object $z^i_l$ and an unlabeled object $z^j_u$, if they belong to the same class $c_m$, the probability that $\|d_{ij}\| < \epsilon$ is:

$$P(\|d_{ij}\| < \epsilon \ \big|\ c(i=j)) = \frac{1}{\sqrt{4\pi}} e^{-\frac{1}{4}\|\mu_{l,c_m} - \mu_{u,c_m}\|^2} = \frac{1}{\sqrt{4\pi}} \tag{29}$$

If the two objects belong to two different classes $c_m$ and $c_n$, then the probability that $\|d_{ij}\| < \epsilon$ is,

$$P(\|d_{ij}\| < \epsilon \,\big|c(i \neq j)) = \frac{1}{\sqrt{4\pi}} e^{-\frac{1}{4}\|\mu_{l,c_m} - \mu_{u,c_n}\|^2} \leq \frac{1}{\sqrt{4\pi}} e^{-\frac{\lambda^2}{4}} \tag{30}$$

Based on the Naive Bayesian rule, when $\|d_{ij}\| < \epsilon$, then the posterior probability that

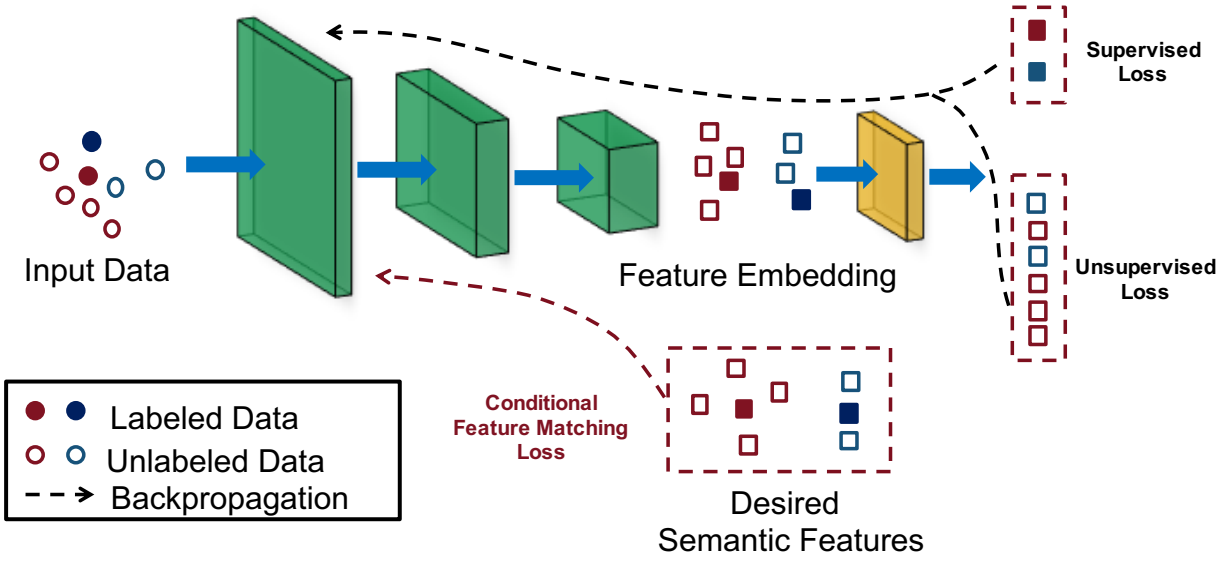



Figure 11: Training Process of Conditional Feature Matching

$z_l^i$ and $z_u^j$ belong to the same class is,

$$\begin{aligned} P(c(i=j)\big|\|d_{ij}\|<\epsilon) &= \frac{P(\|d_{ij}\|<\epsilon|c(i=j))P_{ij}^+}{P(\|d_{ij}\|<\epsilon)} \\ &= \frac{P(\|d_{ij}\|<\epsilon|i=j)P_{ij}^+}{P(\|d_{ij}\|<\epsilon|c(i=j))P_{ij}^+ + \sum_{c_m\neq c_n} P(\|d_{ij}\|<\epsilon|c(i\neq j))P_{ij}^-} \\ &\geq \frac{P_{ij}^+}{P_{ij}^+ + \sum_{c(i\neq j)} P_{ij}^- e^{-\frac{\lambda^2}{4}}} \end{aligned} \tag{31}$$

Lemma 3 is proven. □

Note $0 \leq P_{ij}^- \leq 1$. Because the parameter $\lambda$ is large, $e^{-\frac{\lambda^2}{4}}$ is close to 0. Therefore, typically $P(c(i=j)\big|\|d_{ij}\|<\epsilon)$ is close to 1. This shows that, if a pair of labeled and unlabeled objects are close, then the probability that they belong to the same class is close to 1.

**The Connection to Covariate-shift Condition.** By Lemma 3 if $\|z_l^i - z_u^j\| < \epsilon$, then the probability that they belong to the same class is close to 1. That is, $P(c(i=j)\big|\|z_l^i - z_u^j\| < \epsilon) = 1$. This is equivalent to $p_l(y|z) = p_u(y|z+\epsilon\cdot r)$, where $r$ represents any unit vector. When $\epsilon$ approaches to 0, then $(z+\epsilon\cdot r)\to z$. Therefor, the Covariate-shift condition $p_u(y|z) = p_l(y|z)$ holds.

## 8.2 The Feature Embedding Method

Because Eq. 27 is differentiable, the conditional feature matching loss can be seamlessly plugged into the loss function of the semi-supervised classification model $\mathcal{F}(x,\theta)$, re-

quiring no change to its learning strategy. Therefore, we can leverage any existing semi-supervised model to learn a semantic feature space satisfying the Covariate-shift condition. In this work, we adopt the state-of-the-art of semi-supervised classification, namely, Generative Adversarial Network (GAN)-based SemiGAN model [85, 22].

**An Overview of SemiGAN.** SemiGAN is an extension of the Generative Adversarial Networks(GAN) [66, 75]. GAN is a model originally developed to generate synthetic data objects. Toward this goal, GAN model trains a Generator network $G$ to transform a random vector $z \sim \mathcal{N}(0, I)$ to a synthetic data objects. To evaluate the quality of generated objects, GAN contains a discriminator network $D$ that is trained to distinguish the true data objects and synthetic data objects. The generator network is then trained to fool the discriminator network to accept its output as being real. In turn the discriminator network needs to learn a precise boundary of true data objects to reject the fake objects. In the learning process, GAN produces a good feature embedding of the true data.

Same to GAN, SemiGAN also has a generator $G$ and a discriminator $D$. Unlike the traditional GAN, in the SemiGAN model, the discriminator and generator use different objective functions in order to leverage a small number of available labels. The generator aims to generate fake objects that have a similar feature embedding to that of real objects in the semantic feature space. The discriminator is jointly trained to satisfy two objectives, namely correctly classifying the labeled objects and distinguishing the fake objects from real objects.

Formally, the objective function of the generator in SemiGAN is:

$$loss_G = ||\mathbf{E}_x(\mathcal{M}(x)) - \mathbf{E}_s(\mathcal{M}(G(s)))||_2^2 \tag{32}$$

where $\mathcal{M}$ is the feature embedding model of the discriminator. $s$ is the input of the generator model and sampled from a Gaussian distribution. $G(s)$ is the fake data object produced by the generator.

In a $K$ class classification task, the objective function of the discriminator in SemiGAN is composed of two types of losses, namely, the labeled loss to classify the real objects:

$$loss_l = \mathbf{E}_{x,y\in\mathcal{D}_l}(logP_D(y|x, y \leq K)) \tag{33}$$

and the unlabeled loss that distinguishes the real and fake objects:

$$loss_u = \mathbf{E}_{x,y\in\mathcal{D}_u}(logP_D(y \leq K|x)) + E_s(logP_D(K+1|G(s))) \tag{34}$$

By Eq. 34, the discriminator classifies the unlabeled real objects into one of the $K$ classes, while assigns the fake objects to a $K+1$th class.

**Enhancing SemiGAN with CFM.** LANCET learns a semantic feature space that satisfies the Covariate-shift condition by adding the conditional feature matching loss $loss_{cfm}$ (Eq. 27) into the objective of the discriminator, as depicted in Fig. 11.

$$loss_D = loss_l + loss_u + loss_{cfm} \tag{35}$$

Because the conditional feature matching $loss_{cfm}$ is differentiable, the optimization strategy of SemiGAN is still effective at minimizing the new form of loss function $l_D$ without further change.

Note LANCET does not rely on any specific semi-supervised models to extract features. Rather, users can choose a semi-supervised model that best fits their targeted classification task and their dataset. Our conditional feature matching loss can seamlessly be added into its loss function.

# 9 Label Propagation

Next, we introduce our label propagation strategy that automatically propagates labels from manually labeled objects to the unlabeled data objects. As described in Sec. 8, after the semantic feature embedding, objects belonging to the same class are close to each other in the semantic feature space. In this scenario, label propagation seems straightforward. For example, we could first measure the similarity between the labeled object $z_l$ and the unlabeled object $z_u$, and then if they were close enough we could propagate the label of $z_l$ to $z_u$. However, this intuitive strategy raises some critical research questions:

(1) How to decide if two objects are similar? The semantic feature space typically corresponds to a high dimensional space, especially when the original input data is complex. Yet effectively measuring the similarity in high dimensional spaces remains an open problem due to the curse of dimensionality [67].

(2) How close is close enough? We need an appropriate similarity threshold to determine if two objects are close enough to be said to belong to the same class. This threshold is hard to set.

**A Linear Model-based Solution.** LANCET uses a lightweight machine learning model to solve the above problems. It fully explores the advantage of the semantic feature space $\mathcal{Z}$ produced by our conditional feature matching (CFM) strategy in that it satisfies the Covariate-shift condition (Def. 7.1). Instead of explicitly measuring the similarity among the objects and carefully selecting a similarity threshold to propagate labels, LANCET learns a machine learning model $\mathcal{F}(z)$ to propagate labels in the semantic feature space $\mathcal{Z}$. This model $\mathcal{F}(z)$ produces a prediction for each unlabeled object given its features in $\mathcal{Z}$. Because $\mathcal{Z}$ satisfies the Covariate-shift condition, $\mathcal{F}(z)$ can be very lightweight. We will prove that in fact a linear neural net [59] is sufficient for this label propagation task.

More precisely, we parameterize the linear neural network $\mathcal{F}(z)$ by a weight matrix $W$ and a bias vector $b$, i.e., $\mathcal{F}(z|W,b) = W \cdot z + b$. We use cross-entropy as the loss function. Given an unlabeled object $z_u$, $\mathcal{F}(z|W,b)$ produces a label vector $\hat{y}_u = [y_1, ...., y_C]$, where $C$ represents the number of classes. Each dimension of $\hat{y}_u$ represents the probability that $z_u$ belongs to one particular class i $\in \{1, ..., C\}$. In deep learning, this vector is typically produced by a softmax function $f(\cdot)$. Learning this linear neural network is straightforward, and we use the common approach of SGD optimization [39].

**Theoretical Guarantee.** We formally prove that this lightweight model is effective at propagating labels, leveraging that the semantic feature space satisfies the Covariate-shift condition (Eq. 25).

**Lemma 4.** Given a linear neural network $\mathcal{F}(z|W,b)$ parameterized by $W$ and $b$, with the softmax activation function $f$, if a labeled object $z_l$ and an unlabeled object $z_u$ satisfy that $||z_u - z_l|| < \delta$, then $||\hat{y}_u - y_l|| \leq \| W\| \cdot \sqrt{C} \cdot \delta$.

*Proof.* Given the softmax function $f(\cdot)$, we get Eq. 36 by [88]:

$$||f(\mathcal{F}(z_u)) - f(\mathcal{F}(z_l))|| < ||J||_F^* ||\mathcal{F}(z_u) - \mathcal{F}(z_l)|| \tag{36}$$

where $||J||^* = \max||J||$, and $||J||$ is the Jacobian matrix of the softmax function. Given a vector $v$, we denote $v_i$ as the $i$th element of $v$, and $f(v)_i$ as the softmax value of $v$ corresponding to class $i$. Then the softmax function and the Frobenius norm of its Jacobian matrix are computed as follows [88]:

$$f(v)_i = \frac{e^{v_i}}{\sum_{j=1}^{C} e^{v_j}}$$

$$||J(f)||_F = \sqrt{\sum_{i=1}^{C}\sum_{j=1,i\neq j}^{C} f(v)_i^2 f(v)_j^2 + \sum_{i=1}^{C} f(v)_i^2(1-f(v)_i)^2}$$

Note $\sum_i f(v)_i = 1$, then $(1-f(v)_i)^2 = (\sum_{j\neq i} f(v)_j)^2$. By Cauchy inequality, $(\sum_{i=1}^{N} a_i)^2 \leq N\sum_{i=1}^{N} a_i^2$. And since $\forall i$, $0 \leq f(v)_i \leq 1$ and $\sum_i f(v)_i = 1$, we have $(\sum_{i=1}^{N} f(v)_i)^2 \leq 1$. Thus we have,

$$\begin{aligned} ||J(f)||_F &= \sqrt{\sum_{i=1}^{C}\sum_{j=1,i\neq j}^{C} f(v)_i^2 f(v)_j^2 + \sum_{i=1}^{C} f(v)_i^2(\sum_{j\neq i} f(v)_j)^2} \\ &\leq \sqrt{\sum_{i=1}^{C} f(v)_i^2[\sum_{j=1,i\neq j}^{C} f(v)_j^2 + (C-1)\sum_{j=1,i\neq j}^{C} f(v)_j^2]} \\ &\leq \sqrt{C\sum_{i=1}^{C} f(v)_i^2} \leq \sqrt{C} \end{aligned} \tag{37}$$

Next, we give the upper bound of $||\mathcal{F}(z_u) - \mathcal{F}(z_l)||$. Since $||z_u - z_l|| < \delta$, from Eq. [88], we have,

$$||\mathcal{F}(z_u) - \mathcal{F}(z_l)|| \leq \|W\| \cdot \delta \tag{38}$$

Combining Eq. 37 and 38, we get $\hat{y}_u - \hat{y}_l \leq \sqrt{C} \cdot \|W\| \cdot \delta$. Since $\hat{y}_l = y_l$, we have $\hat{y}_u - y_l \leq \sqrt{C} \cdot \|W\| \cdot \delta$. Lemma 4 is proven. □

By Lemma 4, with high probability this model will correctly propagate the labels of $z_l$ to the unlabeled objects $z_u$ when $||W||$ is small. Note that we can always learn a small $||W||$ by applying some regularization to the objective function of the neural net [39].

**Jointly Learning with Conditional Feature Matching.** Due to the linearity of this model, it is jointly learnable with the semantic feature embedding model. For example, if LANCET uses SemiGAN to realize our CFM strategy to produce the semantic feature space, then our linear neural network can thus be simply implemented as the final layer of the discriminator of SemiGAN. Therefore, LANCET can learn the semantic feature extraction model and the label propagation model jointly using Stochastic Gradient Decent (SGD) and back propagation, following the typical training process of SemiGAN.

# 10 Label Candidate Selection & Labeling Termination

In this section, we first introduce our label candidate selection method. Then in Sec 10.4 we show that this method naturally establishes an effective criteria for termination of the labeling process.

**Continuity Condition Driven Label Candidate Selection**. Guided by the Continuity condition in Sec. 7, LANCET selects (at most) $b$ objects that, if labeled by the human annotators, would be most effective at improving the quality of the produced labels, where $b$ is a user configurable parameter.

Our label candidate selection method is inspired by the Continuity condition. To recap, the Continuity condition holds as long as the Radon-Nikodym derivative (RND) $\beta(z) = \frac{p_u(z)}{p_l(z)}$ is bounded by a small value (Sec. 7). Here $p_u(z)$ and $p_l(z)$ represent the probability density functions (PDF) of unlabeled objects and labeled objects, respectively. Intuitively, given an unlabeled object with its value as $z$, if the RND $\beta(z)$ is large, then it does not have a close labeled neighbor to represent it.

We thus conclude that to satisfy the Continuity condition, we should give high priority to the unlabeled objects with large RND when selecting objects for labeling, so that unlabeled points have sufficient numbers of labeled points near-by to perform propagation. Therefore, once we have computed the RND for each object, naturally we can select as labeling candidates the $b$ objects that have the *largest* RNDs.

However, calculating the RND requires the estimation of the PDF $p_u(z)$ and $p_l(z)$. This is hard in our scenario where the dimensionality of the semantic feature space often is high, because accurately estimating the PDF of high dimension data is an open problem [91, 119, 22, 27].

## 10.1 Learning RND By Distribution Matching

To solve this problem, we propose a method that directly estimates the RND $\beta(z)$ without having to first separately estimate $p_l(z)$ and $p_u(z)$. For the ease of presentation, we define the concept of weight **w(z)** $= \frac{p_l(z)}{p_u(z)}$ as the reverse of $\beta(z)$. Essentially, $w(z)$ represents the probability that a unlabeled object has a close labeled neighbor.

**Solution: Mapping to the Weighted Distribution Matching Problem.** Our insight is that the problem of estimating the $w(z)$ can be transformed into the *weighted distribution matching* problem.

$$w(z) = \frac{1}{\beta(z)} = \frac{p_l(z)}{p_u(z)} \Rightarrow w(z)p_u(z) = p_l(z) \tag{39}$$

As shown in Eq. 39, after being weighted by $w(z)$, the probability density $p_u(z)$ of the unlabeled objects is equivalent to the probability density $p_l(z)$ of the labeled object. This implies that we can directly estimate $w(z)$ by identifying an appropriate weight for the unlabeled objects such that the weighted distribution of unlabeled objects matches the distribution of labeled objects.

However, even learning $w(z)$ by distribution matching is challenging. First, given two distributions in a high dimensional space, we need a method to evaluate if these two distributions match with each other. Second, even if we can correctly evaluate if the weighted distribution of the unlabeled objects matches the distribution of the labeled objects, we still need an effective yet efficient method to search for the weight $w(z)$ to match the distribution.

We propose to use a *distribution matching network (DMN)* to solve this problem. We first introduce the concept of a DMN in Sec. 10.2 and then show how to efficiently learn these weights through DMN in Sec. 10.3.

## 10.2 Distribution Matching Network (DMN)

Let $\phi(z, \theta_d)$ denote a neural network model, where $\theta_d$ represents the parameters. We use $y_d$ to denote the label of $z$, indicating which distribution $z$ belongs to. The objective is to assign each object to one distribution from which it is most likely sampled. Therefore, we call this neural network the *Distribution Matching Network* (DMN). If two distributions exactly match each other – meaning $p_l(z) = p_u(z)$, DMN will fail to distinguish between the labeled and unlabeled objects in the semantic feature space $\mathcal{Z}$. As a result, its classification errors would be around 0.5 on average. Formally, a DMN $\phi(z, \theta_d)$ minimizes the weighted expected loss between the predicted label $\hat{y_d} = \frac{e^{\phi(z,\theta_d)}}{\sum_{i=0}^{2} e^{\phi(z,\theta_d)_i}}$ and $y_d$, as defined below.

$$\begin{aligned}\mathbf{E}(loss(\hat{y_d}, y_d)) = &-\int_z [w(z)p_u(z)p(y_d)logp(\hat{y_d}|z_u)+ \\ &p_l(z)p(y_d)logp(\hat{y_d}|z_l)]dz \\ &= \frac{1}{(N_u + N_L)}[\sum_{i=0}^{N_u} w_i loss(\hat{y_{di}}, y_d) + \sum_{i=0}^{N_l} loss(\hat{y_{di}}, y_d)]\end{aligned} \tag{40}$$

$$y_d|_{z_i} = \begin{cases} 0, \text{if } z_i \text{ is unlabeled} \\ 1, \text{if } z_i \text{ is labeled} \end{cases} \tag{41}$$

In Eq. 40, $w_i$ denotes the weight corresponding to $z_i$. $w_i$ is given beforehand and remains fixed throughout the training process. The expected loss $E(loss(z, y_d))$ corresponds to the $\mathcal{H}$ divergence, a widely used metric to measure the difference between two distributions [32, 11, 33]. DMN learns the parameter $\theta_d$ to minimize the $\mathcal{H}$ divergence.

If $E(loss(z, y_d))$ is still large after the DMN converges, then the weighted distribution of the unlabeled objects matches the distribution of the labeled objects. In particular, when the two distributions are identical, the expected loss $loss(z, y_d, w(z))$ should be around 1.38 [91], which corresponds to the maximal value of cross-entropy loss in binary classification. This offers us a criteria to verify if two given distributions match each other.

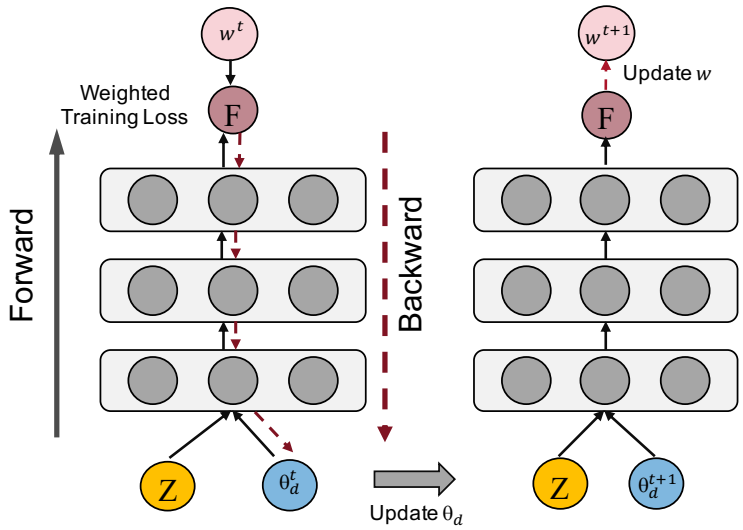



Figure 12: Training Process of Online Weight Approximation

## 10.3 Learning Weights Through DMN: Online Weight Approximation

**Key Idea.** DMN offers us a tool to effectively verify if the weighed distribution of the unlabeled objects matches the distribution of the labeled objects, assuming the weights $w_i$ were to be available beforehand. Next, we show how to use the DMN to learn these weights. The key idea is to treat the weights $\{w_i\}_{i=0}^{N_u}$ as the hyper-parameters of the DMN and learn them by *maximizing* the expected loss of $\theta_d(z, y_d)$. The intuition is that the appropriate weights should best match the distribution of unlabeled objects to that of labeled objects, while two matching distributions will lead to a *large* loss in DMN, as discussed above. This objective is formalized in Eq. 42.

$$\begin{aligned} \theta^*(w) &= \arg\min \mathbf{E}(loss(\hat{y}, y_d)|_w) \\ w^* &= \arg\max \sum_{i=0}^{N_u} w_i loss(\hat{y}, y_d)|_{\theta(w)} \end{aligned} \tag{42}$$

Here $\theta^*(w)$ indicates the learned parameter of DMN given the current weights.

Learning the optimal value of $w_i$ can be very expensive, because the re-weighting process requires two loops of learning, namely, (1) learning the parameter $\theta^*$ of DMN to minimize the loss in Equation 42, and (2) learning the weights to maximize the loss of DMN based on the newly learned parameter $\theta^*$. The two loops iterate back and forth until convergence – until the loss of DMN is around 1.38, because by [91] it indicates the two distributions now match with each other [91].

**Online Weight Approximation: an Efficient Method.** We propose an online weight approximation (OWA) strategy to efficiently learn $w_i$. OWA alternates updates of $\theta^*$ with updates of $w^*$, as shown in Fig. 12. Because the number of unlabeled object $N_u$ can be

large, training a DMN by iteratively learning the weights with respect to all objects is very expensive. OWA employs a mini-batch based optimization strategy to address the efficiency concern. During each training iteration, OWA randomly divides the data into many mini-batches and concurrently learns the weights with respect to each mini-batch, so called *online weight approximation*. Each mini-batch contains only $n \ll N_u$ objects.

More specifically, given a DMN $\phi(z, \theta_d)$, at the $t$-th iteration of the training, $\phi()$ is trained to minimize the weighted loss of the current batch with size $2n$, which includes $n$ labeled data objects and $n$ unlabeled data objects, assuming each $w_i^t$ is fixed at the current iteration $t$.

$$\begin{aligned} loss(\hat{y_d}, y_d)_t &= \frac{1}{2n}[\sum_{i=0}^{n} w_i^t loss(\hat{y_{d_u}}, y_{d_u}) + \sum_{i=0}^{n} loss(\hat{y_{d_l}}, y_{d_l})] \\ \theta_d^{t+1} &= \theta_d^t - \eta \nabla \sum_{i}^{2n} loss(\hat{y_d}, y_d)|\theta_d^t \end{aligned} \tag{43}$$

where $\eta$ is the descent step size of the optimizer.

Then OWA searches for the optimal $w^*$ of the unlabeled objects in the mini-batch that maximizes the weighted loss $loss(\hat{y_d}, y_d)$ by Eq. 43.

$$w^* = \arg\max_{w} \frac{1}{n} \sum_{i=1}^{n} loss(\hat{y_d}, y_d)|_{\theta_{t+1}} \tag{44}$$

Because OWA has to estimate the optimal $w$ for every mini-batch, this estimation has to be lightweight. Therefore, we adopt gradient descent to update the weight $w_i^t$ at each iteration $t$. That is, OWA recalculates the loss based on the updated parameters, and then updates the weight of the unlabeled objects according to the updated loss.

$$\begin{aligned} loss(\hat{y_{d_u}}, y_d)_{t+1} &= \frac{1}{n} \sum_{i=0}^{n} w_i^t loss(\hat{y_{d_u}}, y_d)|_{\theta_{t+1}} \\ w_i^{t+1} &= \frac{\partial}{\partial w_i^t} loss(\hat{y_{d_u}}, y_d)_{t+1} \end{aligned} \tag{45}$$

At the end of each iteration, the weights of all objects in each mini-batch have to be normalized to make sure the total sample weights add up to 1. It is also necessary to ensure $w_i \geq 0$ for all $i$, since minimizing the negative training loss can result in unstable behavior. Therefore, we enforce these constraints:

$$(1) w_i^t = \max(0, w_i^t); (2) w_i^t = \frac{w_i^t}{\sum_{i=0}^{N} w_i^t} \tag{46}$$

Eventually OWA trains the DMN based on Eq. 43 and Eq. 45$\sim$ 46. It assigns a small weight to an unlabeled object if it has a small loss as computed in Eq. 43, because a small loss indicates this object is very different from any labeled objects and hence can be easily recognized to be unlabeled by the DMN. An unlabeled object with a small weight is less

likely to have a labeled neighbor and thus violates the Continuity condition defined in Sec. 7. Accordingly, LANCET selects the $b$ objects with the smallest weights for annotators to label.

**Algorithm 2** Online weight approximation(OWA)

**Require:** $z_u, z_l, y_d, \phi_{z,\theta_d}, \eta, T$ ▷ T: number of iterations.
1: $\forall i = 1 \cdots N_u, w_i \leftarrow 1$ ▷ Initialize $w_i$.
2: $t \leftarrow 0$
3: **while** $t < T$ **do**
4: $\hat{y_d} \leftarrow \frac{e^{\phi(z,\theta_d)}}{\sum_{i=0}^{2} e^{\phi(z,\theta_d)_i}}$
5: $l(\hat{y_d}, y_d) \leftarrow y_d log(\hat{y_d})$
6: $L(\hat{y_d}, y_d) \leftarrow = \frac{1}{2n}[\sum_{i=0}^{n} w_i l(\hat{y_{d_u}}, y_d) + \sum_{i=0}^{n} l(\hat{y_{d_l}}, y_d)]$
7: $\theta_d \leftarrow \theta_d - \eta \nabla L(\hat{y_d}, y_d)|\theta_d$ ▷ Update model.
8: $\hat{y_d} \leftarrow \hat{y_d} = \frac{e^{\phi(z,\theta_d)}}{\sum_{i=0}^{2} e^{\phi(z,\theta_d)_i}}$
9: $l(\hat{y_d}, y_d) \leftarrow y_d log(\hat{y_d})$ ▷ Calculate loss.
10: $w_i \leftarrow \max(0, \frac{\partial}{\partial w_i} l(\hat{y_d}, y_d))$ ▷ Update $w_i$.
11: $w_i \leftarrow \frac{w_i}{\sum_{i=0}^{N} w_i}$ ▷ Normalize $w_i$.
12: $t \leftarrow t + 1$
13: **return** $w_i$

The process of online weight approximation is summarized in Algorithm 2. Given as input the labeled objects $z_l$ and the unlabeled objects $z_u$, and a maximal iteration threshold $T$, it outputs the $w_i$ of each unlabeled object. First, the DMN $\phi_{z,\theta_d}$ is optimized for one gradient decent step using the current weights $w_i$ of the unlabeled objects (Lines 4 – 7). It then recomputes the loss based on the updated parameter $\theta_d$ (Lines 8 – 9) and updates the weight $w_i$ using gradient descent (Lines 10 to 11). The process iterates between these two steps until the number of iterations exceeds $T$.

## 10.4 Termination Condition

To save the human annotator's efforts, the label candidate selection process should terminate when labeling more objects would not significantly reduce the errors of the automatically produced labels.

In LANCET, the distribution matching network (DMN) used in our label candidate selection method naturally offers an effective yet simplistic way to terminate the labeling process. Specifically, LANCET will suggest the annotators to terminate the labeling process when the following two conditions hold: (1) when the weighted distribution of the unlabeled objects is similar to the distribution of the labeled objects; (2) when the Radon-Nikodym derivative (RND) $\beta(z) = \frac{p_u(z)}{p_l(z)}$ is bounded by a small value $\beta_t$ and consequently the Continuity condition holds.

Verifying Condition (1) is straightforward. As discussed in Sec. 10.1, when the two distributions are identical, the expected loss $l(z, y_d, w(z))$ (Eq. 40) should be around 1.38 by [91]. Therefore, we can determine if Condition (1) holds simply by checking the value of Eq. 40.

LANCET verifies Condition (2) based on the weight $w_i$ estimated by our online weight approximation (OWA) method for each unlabeled object. By Eq. 39, the weight $w_i$ w.r.t. each unlabeled data $z_i$ corresponds to an approximation of the reverse RND value at $z_i$. Therefore, if $\forall$ unlabeled object $z_i$, $w_i > \alpha_t$ where $\alpha_t = \frac{1}{\beta_t}$, then Condition (2) holds.

Intuitively, when $\alpha_t$ is large (or $\beta_t$ is small), Condition (2) is hard to satisfy. In this case, LANCET is guaranteed to produce a sufficient number of labels to train a robust machine learning model, but potentially wasting some labeling efforts. Based on our experiments, setting $\alpha_t$ around 0.1 or $\beta_t$ around 10 balances the label sufficiency requirement and the labeling costs in all cases. Therefore, we can apply this same threshold to different datasets without careful tuning.

# 11 Related Works

**Weak Supervision.** In recent years, the database community is interested in developing systems and techniques [76, 103, 24] to solve the labeling problem. They aim to use only a small amount of manual labeling to learn good machine learning models. In particular, Snorkel [76] and Snuba [103] use the concept of weak supervision. The goal is to produce high quality labels from a set of labeling sources which produce a large amount of noisy labels. They then produce the final labels by combining the noisy labels using ensemble-based techniques.

Snorkel [76] provides interfaces for users to write labeling functions, which as the labeling sources, produce labels for subsets of data. However, in some scenarios, especially in complex scenarios such as our medical application, designing these labeling functions is hard even for domain experts. Our LANCET instead only requires users to provide some examples for each class, and thus tends to be more user-friendly than Snorkel.

Snuba [103] uses some lightweight machine learning models as labeling sources, such as decision tree or logistic regression, because they are less label thirsty than deep learning and potentially can be trained using a small number of manually supplied labels. However, when applied on complex data such as images and time series, these simplistic models tend to produce poor predictions, as confirmed in our experiments (Sec. 12.2 and 12.3). Therefore, ensemble cannot extract much useful information from them to produce accurate labels. In contrast, because of its feature embedding strategy and the ability of automatically modeling the data distribution in high dimensional space, our LANCET effectively processes complex data.

GOGGLES [24] uses a *transfer learning* inspired method to label image data. Given an input dataset, it first leverages the pre-trained deep learning model for the big ImageNet [25] data to extract features for all instances in the given dataset and calculates the affinity score between each pair of instances. Then, based on this affinity matrix, it infers the labels of the unlabeled images given some manually labeled images. The intuition is that if the instances in the input data resemble some instances of ImageNet, then a fine tune using a small number of labels potentially is sufficient to adapt the pre-trained model to the input data. Therefore, GOGGLES only targets image data. Our LANCET is instead more general and can handle other types of complex data such as time series.

**Active Learning.** Similar to the label selection of our LANCET, active learning aims to construct a small training set that can train a specific machine learning model with satisfactory accuracy. In general, the active learning methods can be divided into two categories. The first category [30, 31, 28, 94] uses metrics to evaluate the importance of the samples to the performance of the current model, such as uncertainty [30], entropy [31] and expected loss [116, 28]. However, it has been shown that such metrics are often hard to estimate in the cutting edge machine learning techniques for example Deep Neural Networks [88]. The second category [88, 109, 94] seeks to form a compact set of unlabeled instances that represents the overall distribution of the entire unlabeled dataset. However, for high dimensional data such as images, it requires a large number of samples to

represent such a distribution. Driven by the Continuity condition, LANCET instead cleverly picks specific samples to label that ensure the unlabeled objects always have near-by labeled neighbors. This maximizes the efficacy of automatic label propagation – a critical feature not considered in active learning.

**Semi-supervised Classification.** Semi-supervised classification leverages the properties of unlabeled data objects to improve the accuracy of machine learning models. Its goal is to classify data as accurately as possible, assuming a small set of labels is available beforehand. Although similar to our work in that it also solves problems caused by a shortage of labeled objects, it tackles classification as a problem rather than the labeling generation problem. Therefore, unlike LANCET, it does not cover the label selection problem, that is, how to select the most informative objects to label.

Some semi-supervised classification techniques [85, 22, 95, 106, 47, 53, 96, 58, 69] use unlabeled data to learn the low dimensional manifold of the dataset. They then train a machine learning model on this low dimensional feature embedding and thus are unlikely to overfit the small labeled dataset. As discussed in Sec. 8.1, our conditional feature matching strategy naturally leverages these techniques to produce feature embeddings that best fit label propagation.

**Deep Learning-based Feature Extraction.** Deep learning models have been proposed to extract features from complex datasets such as images. In particular, Generative Adversarial Net (GAN) [22, 85, 75] extracts features by forcing a neural network to learn a high density manifold distribution to resist the adversarial attacks from another co-trained neural network. Auto-Encoder [47, 53, 106] methods extract features that are most informative for reconstructing the data objects. Some other works instead design customized heuristics for a certain type of datasets, such as data-augmentation [95, 19], rotation prediction [35], relative patch prediction [72] and colorization [125]. Although all these deep learning-based methods can be used by our LANCET to extract features, these methods do not guarantee that the objects belonging to the same class fall into a small region in the learned feature space – essential to LANCET.

# 12 Experiments

We have conducted an experimental study to evaluate the effectiveness of LANCET. Specifically, we focussed on the following three questions:

1. **Quality of Generated Labels**: How does LANCET compare with existing labeling approaches in term of the quality of the generated labels?

2. **Accuracy of Trained Machine Learning Models**: How does the performance of the machine learning models trained with labels generated by LANCET compare with the models trained with the labels generated by existing labeling approach?

3. **Ablation Study**: How effective are the the key techniques of LANCET work relative to each other?

## 12.1 Experiment Setup

**Datasets.** We evaluate our methods on classification tasks using in total four benchmark datasets including two classic image datasets and two popular time series datasets described below.

• ***SVHN*** *Street View House Numbers* dataset [71] is a widely used real-world image dataset obtained from house numbers in Google Street View images. Each image contains a digit ranging from 0 to 9; thus it has 10 classes. SVHN consists of 73257 training images and 26032 images for testing. The resolution of each image is $3\times32\times32$.

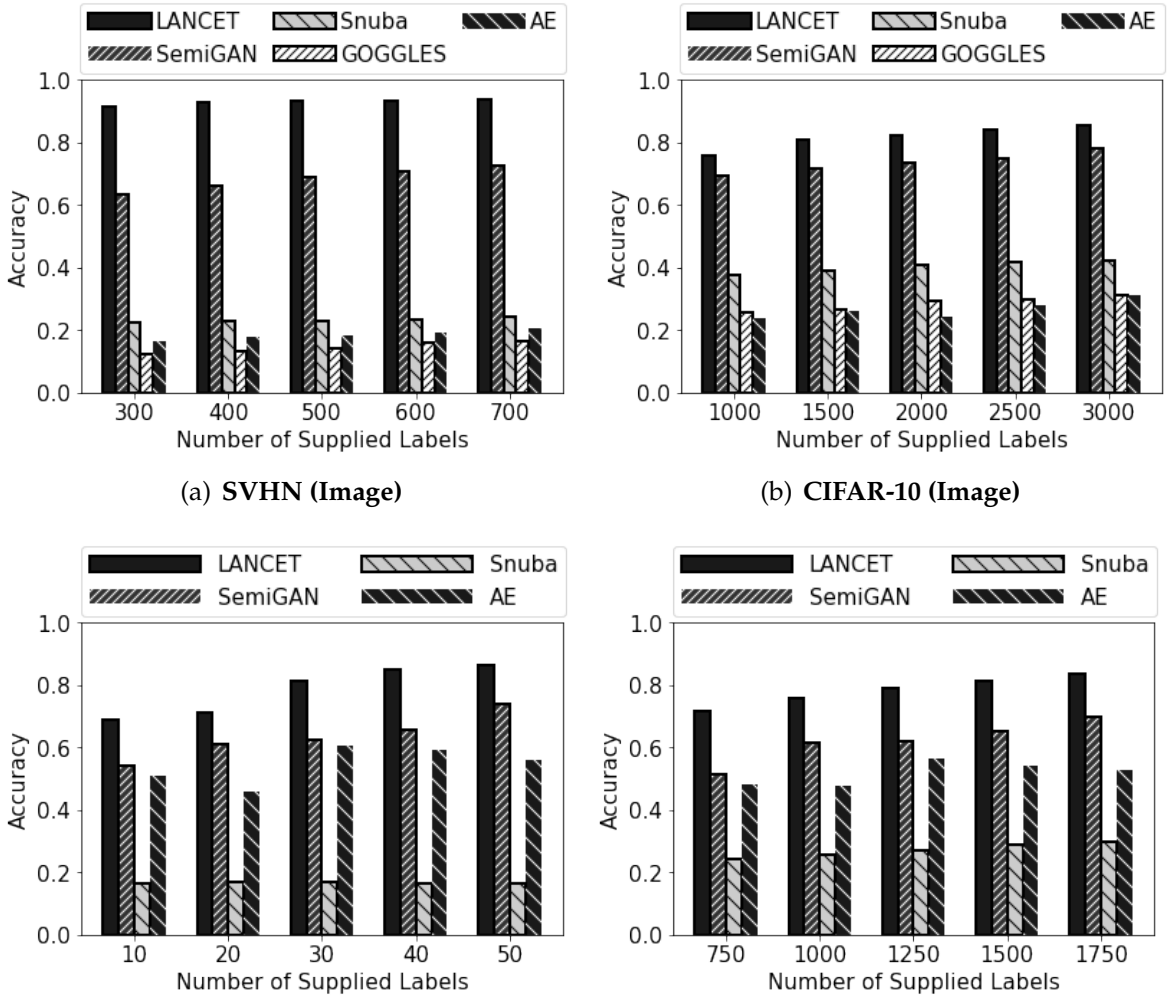
• ***CIFAR-10*** *benchmark* dataset [56] is a popular image dataset composed of 10 classes of natural scenes with 50000 training images and 10000 testing images. Each is an RGB image of size $32\times32$.

• ***HAR*** *Human Activity Recognition* dataset [8] is a popular multivariate time series dataset built from the recordings of 30 subjects performing activities throughout their day while carrying a waist-mounted smartphone with embedded inertial sensors. The activities correspond to 6 classes including Sitting, Standing, Walking, etc. The continuous recordings are broken into 10,299 none-overlapping time segments. Each segment is composed of 128 readings over time with each reading a 9-tuple produced by 9 sensors – thus a $9\times128$ time series.

• ***SpeechCommands*** *[110]* is a public time series dataset used for speech recognition. It has 65,000 utterances of 30 short words by 2,618 speakers. Each utterance is stored as a one-second (or less) WAVE format file encoding the sample data as linear 16-bit single-channel PCM values at a 16 KHz rate. To preprocess the utterances, we first extract normalized spectrograms from the original waveforms and then resize the spectrograms to equalize their sizes at $160\times101$, as in [110].

**Alternative Methods.** We compare LANCET against the core state-of-the-art labeling approaches including Snuba [103], GOGGLES [24], and Core-Set [88]. We also compare LANCET against SemiGAN [22] and AutoEncodoer [106] as additional baselines.

• **Snuba** [103] is the state-of-art of weak supervised label generation system. As the successor of Snorkel [76], Snuba uses a small set of manually labeled examples as seeds




(a) **SVHN (Image)**

(b) **CIFAR-10 (Image)**

(c) **HAR (Time Series)**

(d) **Speechcommands (Time Series)**

Figure 13: Accuracy of Generated Labels: Varying the Number of Manually Supplied Labels.

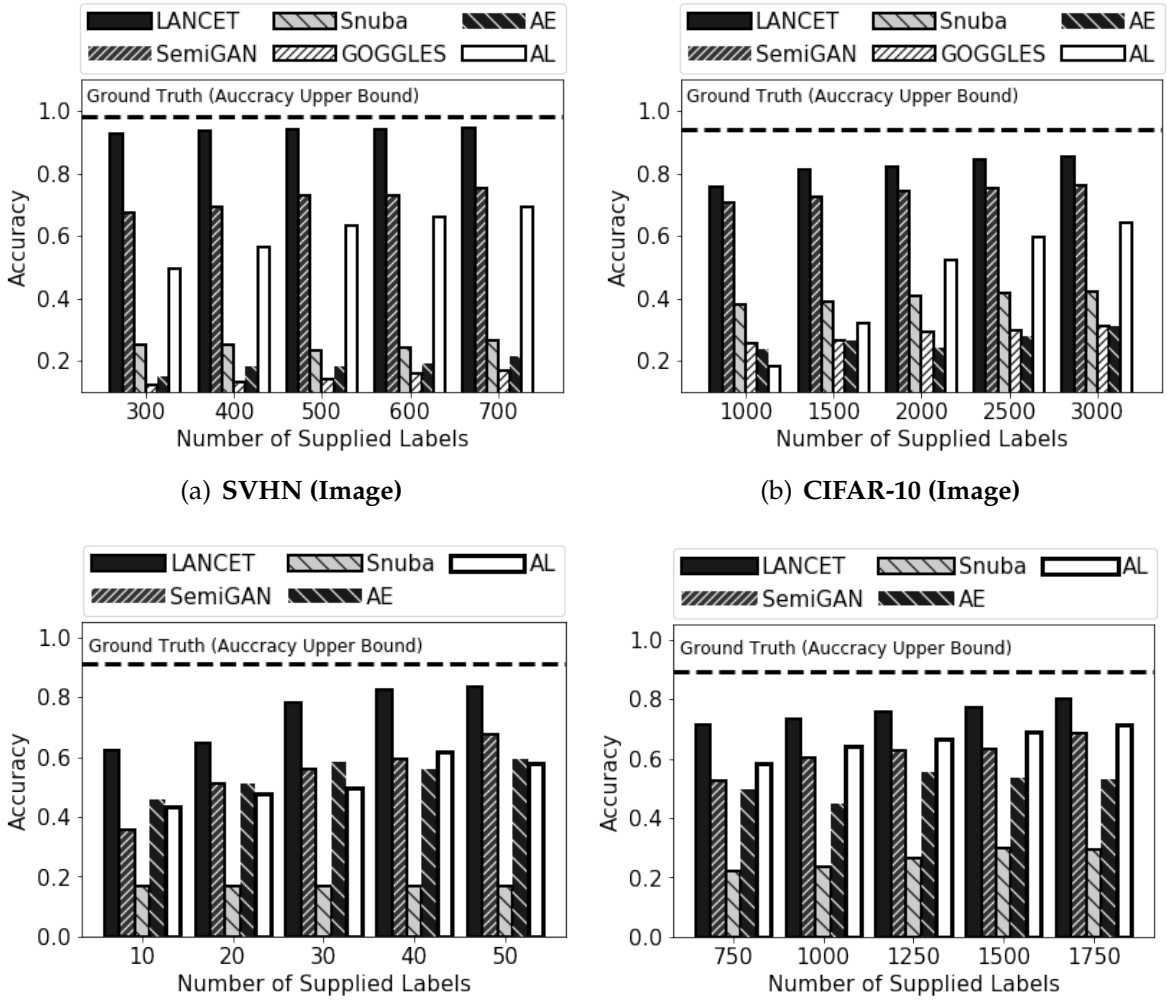
to produce labels.

• **GOGGLES** [24] is the most recent labeling approach targeting on image data. Since GOGGLES does not support time series data, we compare against GOGGLES using the two image datasets, namely SVHN and CIFAR-10. We use the code made available by the authors.

• **Core-Set** [88] corresponds to the state-of-art ~~of~~ active learning designed for deep neural nets which are used in our experiments.

• **SemiGAN** [22] is the state-of-the-art of semi-supervised feature embedding. It is also able to produce a prediction with respect to each unlabeled object in the given dataset.

• **AutoEncoder** [106]. AutoEncoder is a popular feature embedding method. we develop this baseline by replacing the feature embedding component of LANCET with a




(a) **SVHN (Image)**

(b) **CIFAR-10 (Image)**

(c) **HAR (Time Series)**

(d) **Speechcommands (Time Series)**

Figure 14: The Accuracy of the Trained Machine Learning Models: Varying the Number of Manually supplied Labels.

pre-trained AutoEncoder model. This new baseline then uses LANCET's label propagation method to produce labels.

- **Ground Truth.** The machine learning models trained with the *ground truth* labels using the deep neural networks are also evaluated. They represent the *best case upper bound* on the prediction accuracy w.r.t each dataset, as they are given apriori all the correct labels instead of first having to infer these labels.

**Methodology.** We measure the **quality of the produced labels** and the **testing accuracy** of the machine learning models trained on these labels. Following the state-of-the-art [24], in most cases we evaluate LANCET on CNN-based classification problems, because CNNs are known to perform better than other models on complex data such as images or time series.

We also run additional experiments to evaluate the performance of LANCET on other model architectures. In particular, we use Support Vector Machines (SVM) as the downstream machine learning model. We run the experiments on the HAR dataset, because SVMs are known to work well on it.

We measure the quality of the produced labels following the experimental methodology of Snuba and GOGGLES. That is, given a small set of human supplied labels, we use one label generation approach to propagate labels to all unlabeled objects and then evaluate the *accuracy* of the automatically produced labels. Here accuracy is measured as the ratio of the correctly generated labels over all generated labels.

When evaluating the accuracy of the trained machine learning models, in addition to comparing to Snuba, GOGGLES, SemiGAN, and AutoEncoder, we also compare LANCET against the models trained on the labels supplied by the active learning method Core-Set, as well as the models trained using the ground truth labels.

In our ablation study, we evaluate the accuracy of the models trained on the labels produced by different LANCET-based variants to verify the effectiveness of our condition feature mapping (CFM) strategy and the label candidate selection method, which correspond to the key techniques of LANCET. In addition, we also evaluate how active learning works when used together with weak supervision (Snuba + AL).

Finally, we evaluate the termination condition by varying the termination threshold $\alpha_t$ and measuring how the label propagation efficiency of the full-fledged LANCET changes, where propagation efficiency represents the number of *correct* labels automatically inferred per human label.

## 12.2 The Accuracy of Generated Labels

In this set of experiments, we evaluate LANCET, Snuba, SemiGAN, and AutoEncoder on all four image and time series datasets, while GOGGLES was only run on the image datasets, because GOGGLES is specific to image data. When running Snuba on the image datasets, we first use the pre-trained VGG16 model on ImageNet to extract features from the raw image data. We then use principal component analysis (PCA) to project the extracted features into a lower dimensional space with densely rich features, as in the GOGGLES [24] paper. For the time series datasets, Snuba directly uses the raw data as input features to train the models.

All experiments start with a small initial pool of labeled examples randomly sampled from the dataset, corresponding to about 0.5% of the total dataset. The pool of unlabeled data corresponds to the remaining data, from which candidates are selected for the human labelers to annotate. Because all datasets have ground truth available, in our experiments an oracle who already knows the ground truth beforehand simulates human labeler (that is, the oracle will always provide a perfect label when asked). We gradually increase the number of human supplied labels and evaluate the accuracy of the generated labels. We use the same number of human supplied labels for all methods in all experiments.

As shown in Fig. 13, LANCET consistently outperforms other methods on all datasets with a large margin. Next, we explain where the superiority of LANCET comes.

First, our conditional feature matching (CFM) strategy presented in Sec. 8.1 produces a feature embedding satisfying the Covariate-shift condition (Def. 7.1). That is, in the embedding space, the objects belonging to different classes are well separated; while the unlabeled objects tend to share their label with that of their near-by labeled neighbors. We confirm this by visually examining the plot of objects from each dataset by plotting the feature embedding produced by LANCET using t-SNE, a widely used visualization technique that perceives proximity when mapping to 2-dim space to support visual inspection. Because the feature embedding satisfies the Covariate-shift condition, LANCET is able to effectively propagate labels from labeled objects to their unlabeled neighbors using a lightweight linear model, as analyzed in Sec. 9.

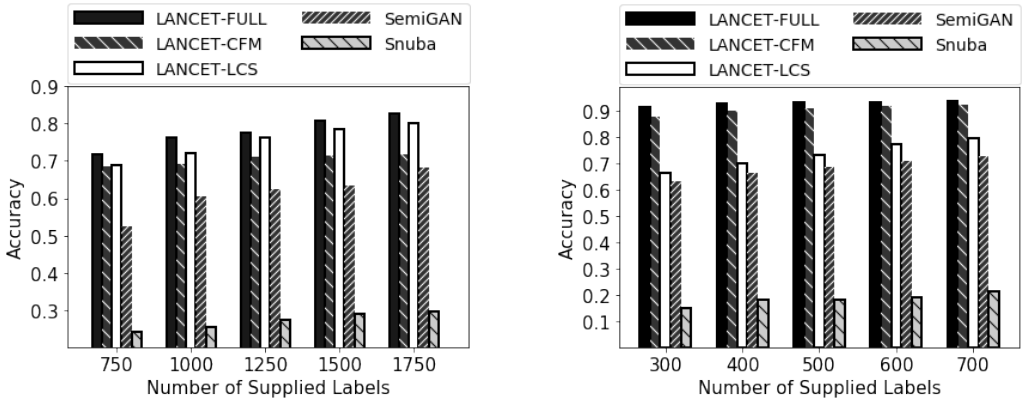
Although SemiGAN also produces a feature embedding that separates objects belonging to different classes, it tends to push the labeled objects far from any unlabeled objects. Thus, it violates the Covariate-shift condition, as depicted in Fig. 10 (a). It is thus ineffective in helping the system to predict the labels of the unlabeled objects, because many of them do not have near-by labeled neighbors to utilize. Therefore, although SemiGAN has been shown to perform better than other baseline methods, especially when handling image datasets, LANCET still consistently outperforms SemiGAN by up to 20 percentage points.

Second, driven by the Continuity condition (Def. 7.2), our distribution matching network-based (DMN) method presented in Sec. 10.3 successfully discovers the areas in the high dimensional embedding space that do not contain enough labeled objects. By selecting objects in these areas for the human domain expert to manually label, LANCET discovers the labeling seeds that are most effective at automatically producing new labels. Therefore, LANCET quickly reaches a high accuracy using very few labels and is able to consistently improve the accuracy of the generated labels as the number of human labels increases, as shown in Fig. 13.

## 12.3 The Accuracy of Trained Models

As was done in Snuba and GOGGLES, we evaluate the accuracy of the machine learning models trained on the automatically generated labels. For all datasets, we use the standard train/test split from the original source.

For the image classification task, all methods use the popular Preact-ResNet [45] as the downstream machine learning architecture, as was done in GOGGLES [24]. For the time-series dataset *Speechcommands*, we designed CNN-based deep neural net to classify them by treating them as image data. Our experimental results in 14(d) show it works well. For the timeseries dataset *HAR*, we use Support Vector Machine (SVM) as the downstream machine learning model, because SVM is known to work well on it. We gradually increase the number of human supplied labels until we meet the termination condition of LANCET, where the termination threshold $\alpha_t$ is set to 0.1.




(a) **Speechcommands (Time Series)** (b) **SVHN (Image)**

Figure 15: Ablation Study

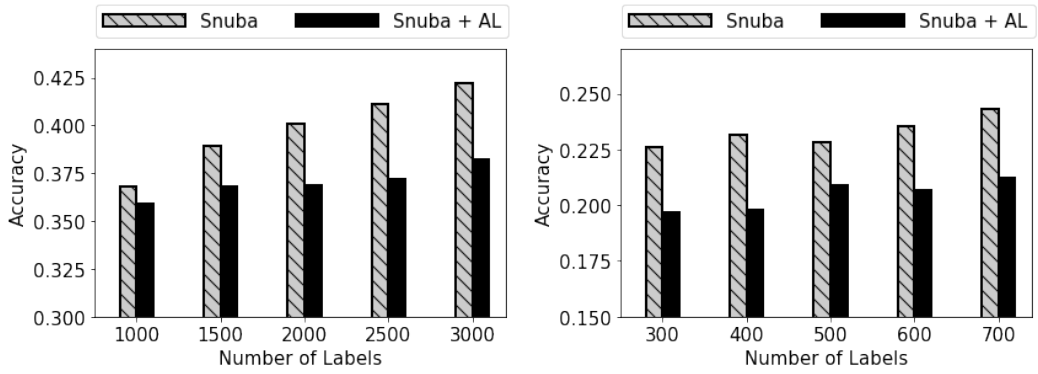
As shown in Fig. 14, the accuracy of LANCET is at least 40 percentage points higher than that of Snuba and GOGGLES. LANCET also significantly outperforms SemiGAN and AutoEncoder by up to 25 percentage points. This is because the labels produced by LANCET are much more accurate, for the reasons described above in Sec. 12.2.

Although the active learning method Core-Set is able to improve the model accuracy as the number of manual labels increases, LANCET still outperforms active learning in all scenarios. This is because LANCET automatically produces a large number of labels with high accuracy, while a much larger training set tends to produce a more accurate machine learning model when the error rate on the labels is low.

In comparison to the fully supervised model trained on the full set of ground truth data, the accuracy of LANCET is only slightly less than the ideal model by 0.05 to 0.1, while using at most 11% of the ground truth labels as human supplied labels.

In particular, in the case of the SVHN dataset (Fig. 14(a)), LANCET performs almost identically to a model trained on the fully labeled dataset, while it only uses 1% of the manually supplied labels. We believe the superior performance of LANCET on SVHN comes from the high quality feature embedding LANCET produces. To confirm this, we use t-SNE to visualize the feature embedding produced by LANCET and by SemiGAN. We find that LANCET's feature embedding component, that effectively enhances SemiGAN with our conditional feature matching (CFM) strategy, is able to produce a high quality feature embedding where objects from different classes are well isolated from one another and the labeled objects share the same label with their near-by labeled neighbors. Thus it satisfies the Covariate-shift condition (Def. 7.1).

The feature embedding produced by SemiGAN also successfully separates the objects belonging to different classes. Therefore, similar to our LANCET, using a small number of manual labels, SemiGAN achieves high accuracy and significantly outperforms other baselines. This indicates that the SemiGAN structure effectively captures the distinct properties of different classes in SVHN and in turn explains the extraordinary performance of LANCET in this case.




(a) **Speechcommands (Time Series)** (b) **SVHN (Image)**

Figure 16: The Accuracy of Snuba with/without Active Learning

## 12.4 Ablation Study

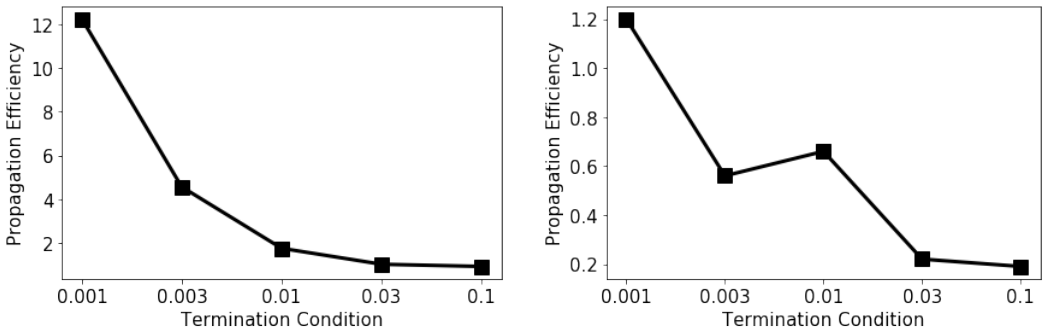
We conduct an ablation study on SpeechCommand and SVHN datasets. We evaluate the effectiveness of our key techniques including the conditional feature matching (CFM) for feature embedding and label candidate selection (LCS) strategies, subject to different types of datasets (image data and time series data). LANCET-CFM uses our CFM strategy for feature embedding, but randomly samples objects for the humans to label (no label candidate selection). LANCET-LCS uses our label candidate selection method to select labeling candidates, but employs SemiGAN [22] to learn feature embedding model (no CFM). We compare the LANCET-based methods against SemiGAN and Snuba.

As shown in Fig. 15, LANCET-LCS outperforms SemiGAN by up to 17 percentage points. Because LANCET-LCS uses the same feature embedding model to SemiGAN, this confirms that the label candidate selection strategy of LANCET is clearly more effective than randomly picking objects for humans to label.

LANCET-CFM significantly outperforms SemiGAN by up to 26 percentage points. Because LANCET-CFM and SemiGAN both randomly picking labeling candidates, this confirms that our CFM strategy effectively improves the quality of feature embedding.

Moreover, LANCET-FULL outperforms LANCET-CFM and LANCET-LCS. This shows that our CFM strategy and label candidate selection strategy are compatible with each other, because they are developed based on one unified theoretical foundation in an integrated fashion.

**Weak Supervision + Active Learning.** This set of experiments also compares Snuba + AL against the original Snuba. The original Snuba uses randomly sampled manual labels as the labeling seeds, while Snuba + AL uses active learning (Core-Set method) to select the manual labels. The results in Fig. 16 show that Snuba + AL performs even worse than the original Snuba, confirming that the active learning methods do not necessarily work well when used together with weak supervision, as noted in the introduction.




(a) **Speechcommands (Time Series)** (b) **CIFAR-10 (Image)**

Figure 17: Termination Condition: Varying Threshold $\alpha_t$

## 12.5 Evaluation of Termination Condition

We evaluate the effectiveness of the termination condition in LANCET. We report the results on CIFAR-10 and Speechcommands datasets. The results on SVHN and HAR datasets show the similar trend.

For this, we vary the termination threshold $\alpha_t$ from 0.001 to 0.1. As analyzed in Sec. 10.4, the larger the $\alpha_t$ is, the more human labels will be collected.

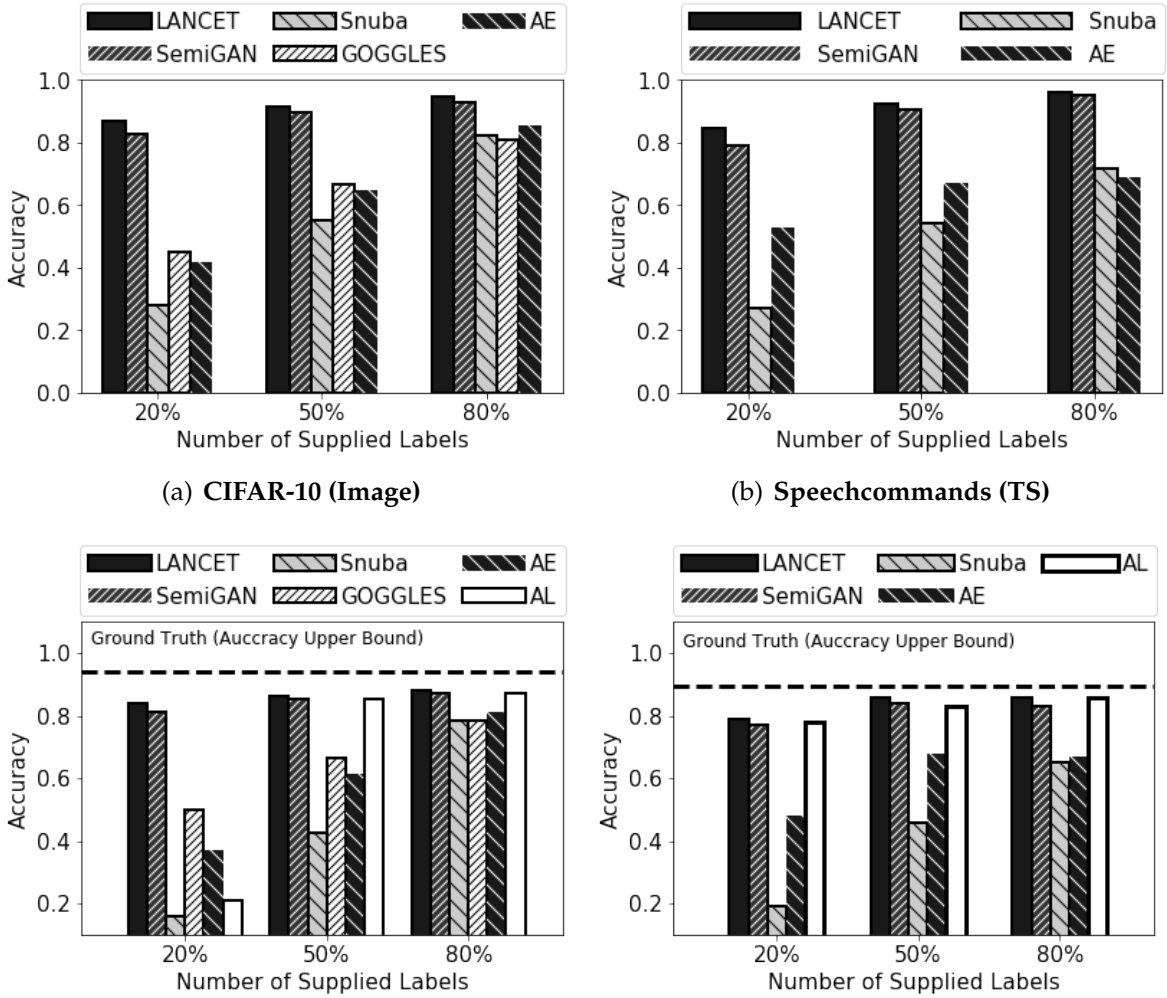
To quantify the effectiveness of the acquired human labels in improving the quality of the automatically produced labels, we define a metric, Propagation Efficiency (PE), which measures the number of *correct* labels automatically inferred per human label. For example, if 500 human labels correctly result in the production of 5,000 labels, then the propagation efficiency (PE) is $\frac{5000}{500} = 10$.

When evaluating the termination condition, after reaching each $\alpha_t$ and where the algorithm would normally terminate, we instead conduct one additional round of labeling and measure the PE based on the new acquired human labels. As shown in Fig. 17, the PE decreases as $\alpha_t$ gets higher. In particular, in both cases when $\alpha_t$ is higher than 0.03, the PE gets stable and drops to below 0.5, indicating two human labels only produce one additional label. This confirms the effectiveness of using $\alpha_t$ as the termination condition. Based on the experiments, we recommend the users to set $\alpha_t$ around 0.03 - 0.1.

## 12.6 Evaluation of Large Proportion of Manual Labels

In this set of experiments, we evaluate how LANCET performs when a large proportion of manually labeled objects are supplied. In particular, we run experiments that use 20%, 50%, and 80% of the objects as manually supplied labels on the CIFAR10 and Speechcommand datasets.

As shown in Fig. 18, all methods perform well when more than 50% of the objects are augmented with manually supplied labels. This is expected, because in these cases, the manually supplied labels often are already sufficient to train a good model. However, LANCET still consistently outperforms other methods even in this case because of its effective feature embedding and label candidate selection strategies, although the perfor-



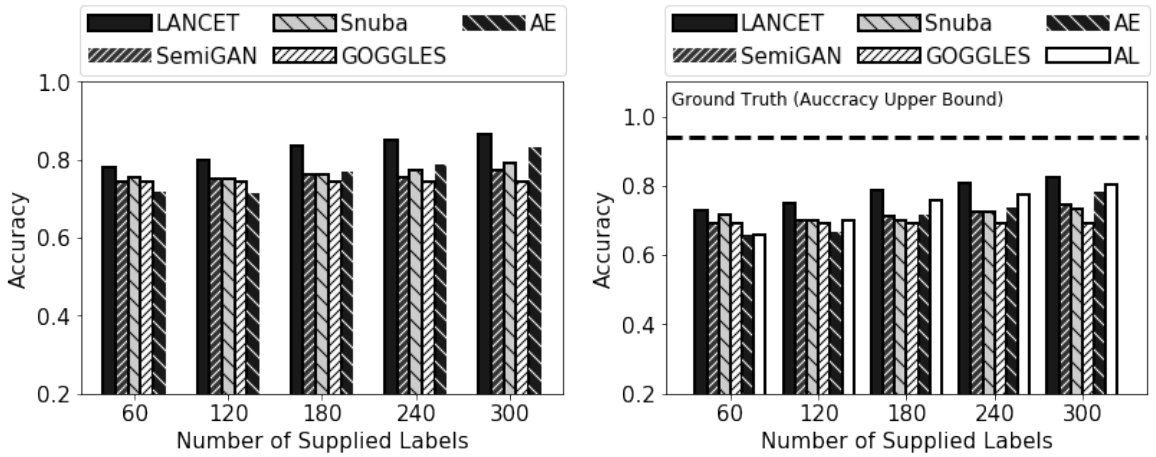

(a) **CIFAR-10 (Image)** (b) **Speechcommands (TS)**

(c) **CIFAR-10 (Image)** (d) **Speechcommands (TS)**

Figure 18: Large Proportion of Manual Labels: (a) Accuracy of generated labels (CIFAR-10). (b) Accuracy of generated labels (Speechcommand). (c) Accuracy of trained models (CIFAR-10). (d) Accuracy of trained models (Speechcommand).

mance gain is not as large compared to the cases where only a small number of manual labels are available.

It is worth noting that LANCET targets automatically producing sufficient labels with *minimal* human labeling efforts. Therefore, we put forth that how LANCET performs under the circumstances with only a small number of manual labels available is much more critical than such scenarios with a large number of manual labels.




(a) **Accuracy of Produced Labels** (b) **Accuracy of Trained Models**

Figure 19: Binary classification on Chest dataset: Varying the Number of Manually supplied Labels.

## 12.7 Evaluation of Binary Classification Task

Because Snuba and GOGGLES report evaluation results for binary classification problems, we conduct experiments to evaluate LANCET on binary classification. To ensure a fair comparison, we run experiments on the *Chest-XRay* [52] dataset used in GOGGLES. This pneumonia chest X-ray dataset contains of 5856 X-ray images labeled by radiologists as being normal or showing different types of pneumonia.

Same as GOGGLES, Snuba, we compare LANCET against all other methods on the accuracy of the produced labels and the downstream classification models. The results show that LANCET consistently outperforms the state-of-the-art methods and other baselines, including GOGGLES, Snuba, SemiGAN, and AutoEncoder, up to 18 percentage points and achieves an accuracy close to 0.9. This shows that LANCET works well in both multi-class classification and in binary classification settings. In this set of binary classification experiments, all methods achieve a relatively high accuracy. A reason for this may be that in binary classification settings, random guess corresponds to an accuracy of 0.5.

# Part III
# MetaStore: Meta Data Analytics for Training Data Curation

### 12.7.1 Proposed Task III: MetaStore: Meta Data Analytics for Training Data Curation

By exploiting the properties of popular DNN models and their gradient computation methodology MetaStore is able to offer an effective solution to these challenges.

***MetaStore Compact Data Storage.*** First, our careful analysis of the back-propagation process of DNN training reveals that the huge gradient of a training sample can be decomposed into 2 small gradients, namely, *prefix* and *suffix* gradients, from which the gradient can be *exactly* re-constructed via a matrix product operation. These two partial-gradients are typically several orders of magnitude smaller than the original gradient especially when produced in layers with a huge number of parameters. Therefore, they are extremely compact, cutting the storage costs from terabytes to gigabytes.

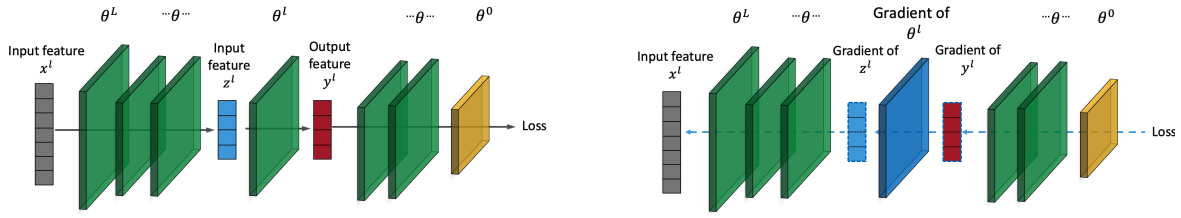
***MetaStore Lightweight Data Collection.*** Instead of first computing the full gradient and then manually decomposing it, we observe that both the small prefix and suffix gradients correspond to intermediate data that could naturally be produced during the back-propagation step when computing the gradient. Their collection can thus be done almost for free, i.e., via a very lightweight process. Better yet, this process is backwards compatible with existing learning processes. For this reason, it can be easily integrated into standard deep learning frameworks, such as PyTorch or Tensorflow, without requiring any system modification. This in turn will improve MetaStore's ease of adoption in practice.

***MetaStore Efficient Analytics.*** MetaStore is the first system to provide *a rich set of operators* that allow users to conduct many gradient-based analytics on the stored metadata from discovering erroneous training samples to interpreting model behavior. These operators often involve computing the inner product similarity of two gradients. This inner product operation tends to impose a significant computational bottleneck [114, 127]. Worst yet, if we were to store two separate prefix and suffix gradients in place of the actual gradient, this would further slow down the inner product operation because MetaStore would have to reconstruct the original gradients each time, before proceeding to execute the specific analytics operations.

In this work, we design an efficient strategy that is able to exactly compute the gradient inner product *without first reconstructing the gradients*. It leverages our observation that given two gradients in a *linear layer*, we can directly use their respective prefix and suffix gradients to compute their inner product via a lightweight linear algebra transformation. With the prefix and suffix gradients much smaller than the gradient itself, this speeds up the inner product operation by several orders of magnitude.

***Generality of the MetaStore Approach.*** The efficient collection, storage, and analytics

services of MetaStore are applicable to all common types of layers (beyond just linear layers) found in popular DNN models such as ResNet [44], VGG [93], and BERT [26]. This holds because all the commonly used layers in these models (each potentially with many trainable parameters and thus producing large gradients), such as the convolutional layers and the self-attention layers, can be decomposed into a set of linear layers.




(a) **Forward Propagation Process** (b) **Backward Propagation Process**

Figure 20: The forward and backward propagation process in Deep Neural Networks.

# 13 Preliminaries

Here, we review the forward and backward propagation processes used for training DNN models as prerequisite to understanding MetaStore's methodology (Fig. 20). A DNN model $\phi(x;\theta)$ is formed by a stack of layers, with $x$ being the input data sample, and $\theta$ being the parameters of the model.

**Forward Propagation Process.** During the inference time, also called forward propagation, the data samples are fed into the first layer. Then each layer takes the previous layer's outputs as its input and transforms the input features into new representations. Finally, the output of the last layer is considered the DNN model's output. The transformation process inside each layer typically is to multiply the input features with a set of parameters, called neurons. Each layer thus can be regarded as a function of the input features and its parameters, i.e.,$z^{l+1} = f^l(z^l, \theta^l)$, where $z^l$ denotes the inputs (output) of the $l$th ($l-1$th) layer, and $\theta^l$ the parameters of the $l^{th}$ layer. The overall DNN model corresponds to a function composition:

$$\hat{y} = \phi(x;\theta) = f^L(f^{L-1}\cdots(f^1(x;\theta^1)\cdots),\theta^L). \tag{47}$$

**Backward Propagation Process.** Deep learning uses backpropagation to train a DNN model. For this, the machine learning practitioners provide the expected outputs of each data sample, such as a label. They also define a loss function that produces a loss value based on the difference between the expected and actual outputs of the DNN model. Then, the gradients of each parameter with respect to the loss value are calculated to update the parameters in the DNN model. More specifically, the gradients of the parameters in each layer are calculated with the chain rule below:

$$\nabla_{\theta^l} C = \frac{dC}{d\theta^l} = \frac{dC}{dz^{l+1}} \cdot \frac{dz^{l+1}}{d\theta^l} = \frac{dC}{dz^L} \cdot \frac{dz^L}{dz^{L-1}} \cdots \frac{dz^{l+1}}{dz^l} \cdot \frac{dz^l}{d\theta^l} \tag{48}$$

where $C$ is the loss value, $C = Loss(\hat{y}, y)$. An optimization method, typically Stochastic Gradient Descent (SGD), updates the parameters $\theta^l$ by taking one step of gradient descent:

$$\theta^l = \theta^l - \alpha \cdot \nabla_{\theta^l} C \tag{49}$$

where $\alpha$ is a predefined learning rate that controls the learning speed of the DNN model.

# 14 Gradient-based DNN Analytics

In this section, we first discuss the foundational principles that most gradient-based DNN analytics techniques are built upon, and then introduce our core operators for gradient-based analytics.

## 14.1 Meta Gradient: The Foundation of Gradient-based Analytics in MetaStore

In deep learning, optimization methods such as SGD directly use gradients to update the parameters of the DNN models. This way, the gradient correlates the training data to the model parameters, and because of this, gradients can be leveraged to measure the contribution of a given training sample to the model performance. This is critical for a range of machine learning tasks, including identifying mis-labeled or dirty training samples, explaining the model's prediction errors, valuating the training samples, and more.

**Observation.** The **meta gradient** – the inner product between the gradients of a training sample and a set of validation samples – effectively estimates how a training sample contributes to the model performance. Below, we theoretically show why this important observation is true.

Intuitively, the contribution of a training sample can be measured by how differently the model would perform if the target sample was not in the training set [77]. Let's consider a standard classification task. The DNN model $\phi(x;\theta)$ is evaluated on a set of validation samples $\{(x_j^v, y_j^v)\}_{j=1}^{N^v}$ that are not in the training set with $y_j^v$ the label of $x_j^v$. We denote the validation loss as $\mathcal{L}^v(\theta) = \frac{1}{N^v}\sum_{j=1}^{N^v} l(x_j^v, y_j^v; \theta)$. Let $\theta_t$ be the parameters of the model that was trained *with* the target training sample $x_t$ and $\theta$ be the parameters trained *without* using the target training sample $x_t$. Then the contribution of the training sample $x_t$ corresponds to the difference between the validation losses of $\phi(x^v;\theta)$ and $\phi(x^v;\theta_t)$, i.e., $L^v(\theta_t) - L^v(\theta)$. With Taylor Expansion, this becomes:

$$L^v(\theta_t) - L^v(\theta) = \nabla_\theta L^v(\theta)(\theta_t - \theta) \tag{50}$$

By Eq. 49, $\theta_t - \theta = \alpha \cdot \nabla_\theta L(\theta)$. Substituting this in, we get:

$$L^v(\theta^t) - L^v(\theta) \propto \nabla_\theta L^v(\theta) \cdot \nabla_\theta L(\theta) \tag{51}$$

In Eq. 51, $\nabla_\theta L^v(\theta) \cdot \nabla_\theta L(\theta)$ represents the inner product between the training example's gradient and the *average gradient* of the validation samples. This is the meta gradient.

Therefore, Eq. 51 substantiates our claim that the meta gradient effectively estimates to what degree a training sample contributes to the model's performance. A positive meta gradient indicates that the training sample impacts the model in a positive way.

## 14.2 MetaStore Gradient-based Analytics

Leveraging the principles of meta gradients, MetaStore provides 4 core operators for gradient-based analytics:

• *Point-to-point (P2P)*: given a training sample and a validation (or, testing) example, estimate the contribution of the training sample to the prediction result of the validation (testing) example.

• *Point-to-batch (P2B)*: given a training sample and a batch of validation (testing) examples, estimate the contribution of the training sample to the prediction results of the batch of validation (testing) examples.

• *Batch-to-point (B2P)*: given a batch of training samples and a validation (testing) example, estimate the contribution of the batch of training samples to the prediction result of the validation (testing) example.

• *Batch-to-batch (B2B)*: given a batch of training samples and a batch of validation (testing) examples, estimate the contribution of the batch of training samples to the prediction results of the batch of validation (testing) examples.

Using these operators as building blocks, for the first time users could easily develop gradient-based analytics techniques to interpret the model prediction by examples [112, 43], debug data issues [14, 74], or valuate the training samples [117, 49], etc. These tasks are critical for deep learning to achieve superior performance. Below are some intuitive examples.

**Interpreting Model Prediction By Examples.** Users could use the P2P operator to first compute the contribution of each training sample to the prediction of one testing sample and then select the top $k$ training samples shown to have the most significant contribution to explain why the model predicts the given testing sample in the identified manner.

**Data Debugging.** Users could use P2B operator to determine how each specific training sample contributes to the prediction of a set of testing samples. If the P2B operator returns a negative value, it indicates this training sample could jeopardize the overall performance of the model. The users thus could identify this sample as a potential outlier or as mislabeled.

**Data Valuation.** Accordingly, using the P2B operator, the users could valuate the training samples based on their contribution to the model. The more the training samples contribute, the more valuable they are. Potentially, the valuation results could guide the users to determine what new training samples they should collect to best improve the model performance.

Similarly, the B2P and B2B operators allow the users to evaluate how a batch of training samples as a whole impacts either the prediction of one testing sample or the overall performance of the model, thus interpreting model prediction or debugging data issues. As deep learning typically updates the model batch by batch using the average gradient of a batch of training samples, these operators mimic the training process of deep learning, thus meaningful.

As input to these operators, the training samples would have already been seen by

MetaStore when training the DNN model. However, MetaStore does not make any assumptions about the testing examples. Instead, it allows the users to specify them at will. Because the gradients produced in a big DNN model tend to be ultra-high dimensional, it is challenging to implement these operators in a way that is both storage and computation efficient.

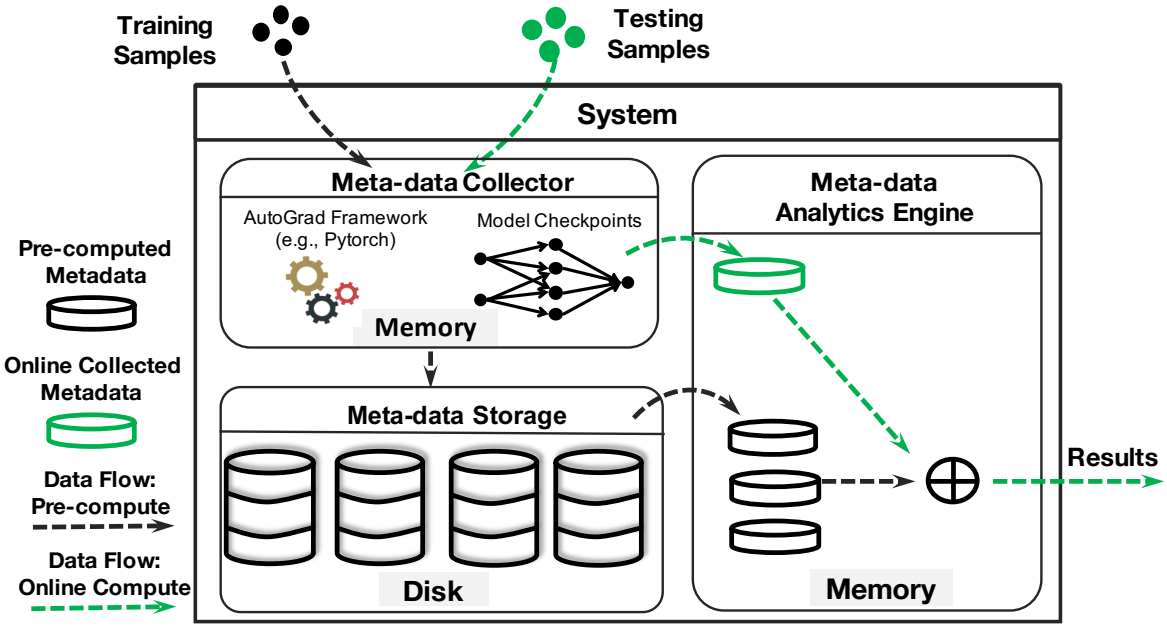



Figure 21: System Overview

# 15 System Overview

Given this background in gradients, in this section we overview MetaStore (Fig. 21), which consists of three key components: (1) a Meta-data Collector, (2) Meta-data Storage, and (3) a Meta-data Analytics Engine.

**Meta-data Collector**: This component continuously collects the meta-data produced in the DNN training process using a non-intrusive process. Because the popular AutoGrad framework (e.g., PyTorch) recently has started to offer interfaces to gain access and materialize the meta-data, MetaStore directly leverages them to collect meta-data. These collected meta-data along with other meta information about the model architectures are then stored.

The Meta-data collector also stores the model $\phi(x;\theta)$ learned at each checkpoint during DNN training. This way, MetaStore is able to collect meta-data for the data samples not seen in the training process, e.g., testing samples. MetaStore achieves this by **model replay**. Given a data sample $x_i$ and a model $\phi(x;\theta)$, model replay first performs a forward propagation process to get the prediction $\hat{y}_i$ of $x_i$ by $\phi(.)$, where $\hat{y}_i = \phi(x_i, \theta)$. It then calculates the loss value $C_i = L(\hat{y}_i, y)$ and performs backward propagation as described in Sec. 13 to obtain the gradient related meta-data. But it does not update the model parameters themselves. By replaying models, MetaStore is able to extract meta-data on the fly at query time. The MetaStore meta-data collector is lightweight and compatible with existing deep learning frameworks with little modification. Thus, we do not discuss it further.

**Meta-data Storage**: The meta-data is maintained on disk (Sec. 16). With a DNN model composed of a series of layers (Sec. 13), a DNN model's gradient equals the concatenation

of each layer's gradient. Thus, in MetaStore, the minimal unit of storage encapsulates the meta data of a specific layer in the DNN, which then is typically stored in a file. If the training set is large, MetaStore may further divide the entire data set into small batches. In this case, each file only contains the meta data corresponding to a small batch of data samples. MetaStore also maintains a directory index that indicates what data samples are stored in which file.

By decomposing the gradient into two partial gradients, namely the prefix and suffix gradients, MetaStore's storage strategies eliminate the storage bottleneck caused by the size of the gradients. The details are discussed in Sec. 16.

**Meta-data Analytics Engine**: This component provides efficient execution strategies for the 4 core operators discussed in Sec. 14.2. The input to each operator is the training and testing samples specified by the users. Because MetaStore already collects the meta-data of all training samples and maintains them in storage, the engine directly loads the requested gradients of the training samples from storage into GPU memory. However, unlike the training samples, MetaStore had not seen the testing samples in the training process. Therefore, it will compute their gradients on the fly by calling the *model replay* function. The engine then efficiently executes these operators using the optimized strategies discussed in Sec. 17 and Sec. 18. In addition, the engine uses caching to maintain the meta-data in GPU memory whenever possible and thus reduces I/O costs. It uses the standard LRU cache replacement policy to evict meta-data when memory overflows [104, 43].

# 16 Space-Efficient Gradient Storage

MetaStore leverages our **prefix/suffix observation** to compactly store the gradients metadata. Specifically we show that storing two small prefix and suffix matrices produced during backpropagation is sufficient to reconstruct the exact original gradient of any data sample. In Sec. 16.1, we introduce the key idea using linear layers as an example layer type. Thereafter, we illustrate that these principles extend to other types of DNN layers, including convolution and self-attention.

## 16.1 Gradient Storage: Linear Layers

Given a DNN model, assume its $l$th layer is a linear layer that applies a linear transformation to the input feature vector $x$.

$$y = \theta \cdot x + B \tag{52}$$

Suppose the input feature vector $x$ and the output feature vector $y$ have $D^{in}$ and $D^{out}$ dimensions, respectively. Then $\boldsymbol{\theta}$ contains $D^{in} \times D^{out}$ parameters. Let $\frac{dC}{d\theta}$ or $\nabla_\theta C$ denote the gradient of this layer. By Eq. 48, $\frac{dC}{d\theta} = \frac{dC}{dy} \cdot \frac{dy}{d\theta}$.

**Prefix Gradient.** The first matrix $\frac{dC}{dy}$ corresponds to the gradient of the output feature vector with respect to the loss value, called *prefix gradient*. Intuitively, it indicates how to update the output features to reduce the loss of the DNN model. Since the loss value $C$ is calculated based on the output of the final layer, calculating the matrix $\frac{dC}{dy}$ requires backpropagation from previous layers. Because the linear layer is the $l$th layer, then $\frac{dC}{dy} = \frac{dC}{dz^L} \cdot \frac{dz^L}{dz^{L-1}} \cdots \frac{dz^{l+1}}{dz^l}$. Although calculating the prefix gradient through backpropagation is expensive, its size is only $D^{out}$. That is, it is identical to the size of the output feature vector $y$, being much smaller than the size ($D^{in} \times D^{out}$) of the final gradient $\frac{dC}{d\theta}$ we are interested in.

**Suffix Gradient.** The other matrix $\frac{dy}{d\theta}$, also called Jacobian matrix, corresponds to the *suffix gradient*. It indicates the expected update on parameters $\theta$ that will produce a better output feature embedding. Even though its general formulation is very complex and large, the Jacobian Matrix in a linear layer is simple:

$$\frac{dy}{d\theta} = \frac{d(\theta \cdot x + B)}{d\theta} = x \tag{53}$$

By Eq. 53, the suffix gradient in the linear layers is in fact identical to its input feature vector $x$. The size, $D_{in}$, is much smaller than the size of the parameters $\theta$.

**Prefix/Suffix Observation.** Naturally, extracting out and maintaining the pair of small prefix and suffix gradients is sufficient to reconstruct the original gradient of a linear layer as follows:

$$(\nabla_\theta C)_{r,s} = (\frac{dC}{dy})_r \cdot \boldsymbol{x}_s \tag{54}$$

where $\nabla_\theta C_{r,s}$ represents the $r$th row and $s$th column of $\theta$.

**Space Complexity.** The space complexity of storing the prefix and suffix gradients is O($D^{out} + D^{in}$), while storing the full gradient takes $D^{out} \times D^{in}$ space. Thus leveraging this prefix/suffix observation, MetaStore drives down the storage costs by $\frac{D^{out} \times D^{in}}{D^{out} + D^{in}}$. As an example, given a $4096 \times 4096$ linear layer commonly used in RestNet or VGG, the savings are 2048-fold.

**Collecting for Free.** Instead of having to first obtain a full gradient and then decompose it into two matrices, the small prefix and suffix gradients correspond to the intermediate data produced during back-propagation when computing the gradient. Collecting them is thus almost free. Better yet, it requires no modification to the standard deep learning frameworks such as PyTorch or Tensorflow.

**General Outlook.** Next, we show that the principle of decomposing gradients into prefix and suffix gradients is also applicable to other typical DNN layers, including those that tend to have a large number of parameters and thus produce huge gradients. This is because: (1) all these layers use the chain rule to compute gradients during backpropagation, and (2) they can each be decomposed into a set of linear layers.

In this paper, we use the convolutional (Sec. 16.2) and self-attention layers (Sec. 16.2) as examples. The convolutional layer is widely used in DNN for computer vision tasks and more recently has been drawing more attention from natural language processing (NLP) and time-series analytics, while self-attention is the core component in any transformer-based architecture. Other similar layers include normalization layers [48], embedding layers, long short term memory (LSTM) layers [34], and gated recurrent units (GRU) [20], to just name a few.

## 16.2 Gradient Storage: Convolutional Layers

For the ease of understanding, we use the standard 1D convolutional layer as an example to illustrate the idea. Same as with the linear layer, we denote the parameters of the convolutional layer as $\theta$. The input data sample $x$ corresponds to a tensor in the shape $(C_{in}, S)$, where $C_{in}$ represents the number of input channels and $S$ the number of features in each channel. For example, if the input data is an RGB image with $32 \times 32$ resolutions, $C_{in}$ is equal to 3 and $S$ equal to $32 \times 32$. Similarly, its output is a tensor with a shape of $(C_{out}, S - K)$, where $C_{out}$ represents the number of output channels and $K$ the number of the dimensions of one kernel. As a 1D matrix, a kernel $\mathcal{K}$ performs the convolution operation on the features of an input channel as follows:

$$y_s = \sum_{i}^{K} \mathcal{K}_i \cdot x_{s+i} \tag{55}$$

The convolution operation produces the output features with $S - K$ dimensions for each individual input channel. Aggregating these output features produces the final features of one output channel $m$:

$$y_{m,s} = \sum_{i}^{C_{in}} \sum_{j}^{K} \theta_{m,i,j} \cdot x_{C_{in},s+j} \tag{56}$$

Repeating this process $C_{out}$ times produces an output with $C_{out}$ channels. Thus, there are $C_{out} \times C_{in}$ kernels in a convolutional layer. In the training process, DNN learns these kernels to produce good output features. Thus, in the convolutional layer, parameters $\theta$ is a tensor with the shape of $(C_{out}, C_{in}, K)$. The final output $y$ is a tensor that contains $C_{out}$ channels, with each channel composed of $S - K$ features.

Similar as with linear layers, the gradients $\frac{dC}{d\theta}$ of the convolutional layer can be decomposed into the prefix gradient $\frac{dC}{dy}$ and suffix gradient $\frac{dy}{d\theta}$, i.e., $\frac{dC}{d\theta} = \frac{dC}{dy} \cdot \frac{dy}{d\theta}$. This is because all layers in DNN use the same chain rule (Eq. 48) to compute gradients during back-propagation.

**The Storage Strategy.** However, whether decomposing the gradient reduces storage costs or not depends on the sizes of the prefix and suffix gradients. Because $C$ is a scalar, the size of the prefix gradient $\frac{dC}{dy}$ is equal to the number of output features. Next, we analyze the suffix gradient $\frac{dy}{d\theta}$. For this, we establish the *connection between the convolutional layer and the linear layer*, so that MetaStore will be able to adapt the storage strategy for the linear layer to the convolutional layer.

Recall that $\theta$ is a tensor in the shape of $(C_{out}, C_{in}, K)$. It can thus be regarded as an aggregation of $K$ linear sub-layers, with the shape of each sub-layer being $(C_{out}, C_{in})$. For the ease of presentation, we denote the $i$th linear sub-layer $\theta_{(\cdot,\cdot,i)}$ as $\theta_i$, and similarly $x_{(\cdot,s)}$ as $x_s$, and $y_{(\cdot,s)}$ as $y_s$. Then, we have:

$$\frac{dC}{d\theta_i} = [\frac{dC}{dy_0} \cdots \frac{dC}{dy_{s-K}}] \cdot [\frac{dy_0}{d\theta_i} \cdots \frac{dy_{s-K}}{d\theta_i}]^T = \sum_{s=0}^{s-K} \frac{dC}{dy_s} \cdot \frac{dy_s}{d\theta_i} \tag{57}$$

From Eq. 56, we have:

$$\frac{dy_s}{d\theta_i} = \frac{d(\sum_{\tilde{i}}^{K} \theta_{\tilde{i}} \cdot x_{s+\tilde{i}})}{d\theta_i} \tag{58}$$

Since $\frac{d(\theta_{\tilde{i}} \cdot x_{s+\tilde{i}})}{d\theta_i} = 0$ if $\tilde{i} \neq i$, while $\frac{d(\theta_{\tilde{i}} \cdot x_{s+\tilde{i}})}{d\theta_i} = x_{s+i}$ if $\tilde{i} = i$. Finally, we have:

$$\frac{dC}{d\theta_i} = \sum_{s}^{S-K} [\frac{dC}{dy_s} \cdot x_{s+i}] \tag{59}$$

Eq. 59 shows that MetaStore is able to reconstruct the gradients of the convolutional layers in a similar way to those of the linear layers. Therefore, MetaStore only needs to store *the prefix gradient and the features of the input samples*, where the size of the prefix gradient is the same as that of the output features.

**Space Complexity.** Storing the gradients as described above, the space complexity of MetaStore is determined by the size of the input samples and the size of the gradient of the output samples, that is, $S \times (C_{in} + C_{out})$. Storing the original gradient takes $K \times C_{out} \times$

$C_{in}$ space, which is identical to the number of parameters in the convolution kernels. Therefore, when $S \times (C_{in} + C_{out}) < K \times C_{out} \times C_{in}$, MetaStore saves space. This is often true. For example, the last layer of the VGG16 model contains $9 \times 512 \times 512$ parameters, while its input and output features are only $512 \times 1 \times 1$ when training a VGG16 model on CIFAR-10 dataset. In this case, the saving is 4068x. In Sec. 19.2, we verify this with experiments.

But when the number of parameters is not large in the convolutional layer, the gradient is small. In this case directly storing the gradient will not constitute a space bottleneck. Given a convolutional layer, MetaStore uses the above space complexity as cost model to determine when to store the prefix and suffix gradients versus storing the original gradients directly.

## 16.3 Gradient Storage: Self-Attention Layers

Here, we use the sentence classification task as an example to show our gradient storage strategy on the self-attention layers (SAL). The input sample $x$ of a SAL is a tensor with the shape of $(S, H)$, where $S$ denotes the length of the sentence and $H$ the number of hidden features of each word. SAL uses Key-Query-Value to produce attention scores and update feature embeddings, accordingly. More specifically, SAL consists of three sub-layers, the key sub-layer $\theta^k$, the query sub-layer $\theta^q$, and the value sub-layer $\theta^v$. Each sub-layer is a *linear layer*. Given an input sample, each sub-layer performs a linear transformation on all word representations in the sentence and generates three representations for each word, namely $z_k$, $z_q$, $z_v$, as shown in Eq. 60 below.

$$z_{ks} = \theta^k \cdot x_s, \quad z_{qs} = \theta^q \cdot x_s, \quad z_{vs} = \theta^v \cdot x_s \tag{60}$$

Then the final output is $y_s = softmax(z_{ks} \cdot z_{qs}/\sqrt{H}) \cdot z_{vs}$.

**Storage Strategy.** The three sub-layers perform the linear transformation on each word in the sentence. The shape of $x$ is $(S, H)$, while the shapes of $\theta^k$, $\theta^q$, and $\theta^v$ are all $(H, H)$. Therefore, the shapes of $z_k$, $z_q$ and $z_v$ are $(S, H)$. This is equivalent to performing a linear transformation on a batch of $S$ samples, where $S$ is the length of the sentence.

Because in a SAL only the three sub-layers contain parameters, MetaStore handles each sub-layers separately. It then simply concatenates the gradients of each sub-layer to obtain the final gradient of the SAL.

Each input sequence can be modeled as a batch of words. Then given a sub-layer, its gradient is equivalent to the sum of the gradients with respect to a batch of data samples, where a sample corresponds to one word. Then given one data sample $x_s$, because the sub-layer is linear, its gradient with respect to this sub-layer can be decomposed the same way as done by the linear layer (Eq. 54), that is, decomposed to a *prefix gradient* and *input features* $x_s$. Finally, the gradient of each sub-layer can be computed with Eq 61:

$$\begin{aligned}\frac{dC}{d\theta^k} = \sum_s^S \frac{dC}{dz_l^k} \cdot x_s, \quad \frac{dC}{d\theta^q} = \sum_s^S \frac{dC}{dz_l^q} \cdot x_s, \quad \frac{dC}{d\theta^v} \\ = \sum_s^S \frac{dC}{dz_l^v} \cdot x_s\end{aligned} \tag{61}$$

Handling each sub-layer separately, MetaStore only needs to store the prefix gradient per layer and the input features, where the size of the prefix gradient corresponds to the size of output features. MetaStore then is able to restore the original gradients using Eq. 61.

**Space Complexity.** The space complexity of MetaStore is $(3H + H) \times S$. Storing the full gradients takes 3 $\times$ $H$ $\times$ $H$ $\times$ S space. So MetaStore drives down the storage costs by $O(\frac{3 \times H}{4})$x. Given a SAL which produces 128 dimensional feature embeddings (H = 128), the saving would be 96 fold.

# 17 Meta-data Analytics Engine: P2P

Next, we describe MetaStore's strategies that efficiently realize the gradient-based analytics operators described in Sec. 14. We introduce the execution strategy for the P2P operator below, while the P2B operator is covered in Sec. 18. Due to space limitation, we only briefly sketch the B2P and B2B operators in Sec. 18.2.

## 17.1 P2P Operator: Linear Layers

We start with the linear layer as an example, and generalize to other layers thereafter. As shown in Sec. 14.1, MetaStore uses the inner product similarity between the gradients of two data samples to estimate the contribution of a training sample to the prediction result of a testing sample. Because MetaStore stores the compact prefix and suffix gradients instead of the original (often huge) gradients, a straightforward solution would be to restore the gradients first and then to compute the inner product. Obviously, this introduces extra overhead due to having to perform the restore operation.

MetaStore succeeds to compute the inner product of two gradients exactly *without having to restore* them first. More specifically, MetaStore could compute the exact inner product of two gradients by first in parallel computing the inner product on the prefix gradient $\frac{dC}{dy}$ and on the suffix gradient $x$, and thereafter multiplying these two results. Because it directly operates on the small prefix and suffix gradients, it is orders of magnitude faster than storing the original gradients before and then directly computing the inner product. Lemma 5 proves the correctness of this optimized method.

**Lemma 5.** Let's denote two data samples as $x_1$ and $x_2$ and their corresponding outputs of a linear layer $\theta$ as $y_1$ and $y_2$. We denote their loss values as $C_1$ and $C_2$, and thus the gradients as $\nabla_\theta C_1$ and $\cdot\nabla_\theta C_2$. Then Eq. 62 holds.

$$\nabla_\theta C_1 \cdot \nabla_\theta C_2 = [(\frac{dC_1}{dy_1}) \cdot (\frac{dC_2}{dy_2})] \cdot [x_1 \cdot x_2]. \tag{62}$$

*Proof.* From Eq. 54, we have,

$$\begin{aligned}
\nabla_\theta C_1 \cdot \nabla_\theta C_2 &= \sum_{r=0}^{D^{out}} \sum_{s=0}^{D^{in}} (\nabla_\theta C_1)_{r,s} \cdot (\nabla_\theta C_2)_{r,s} \\
&= \sum_{r=0}^{D^{out}} \sum_{s=0}^{D^{in}} (\frac{dC}{dy_1})_r \cdot (x_1)_s \cdot (\frac{dC}{dy_2})_r \cdot (x_2)_s \\
&= [\sum_{r=0}^{D^{out}} (\frac{dC}{dy})_r \cdot (\frac{dC}{dy})_r] \cdot [\sum_{s=0}^{D^{in}} (x_1)_s \cdot (x_2)_s] \\
&= [\frac{dC_1}{dy_1} \cdot \frac{dC_2}{dy_2}] \cdot [x_1 \cdot x_2]
\end{aligned} \tag{63}$$

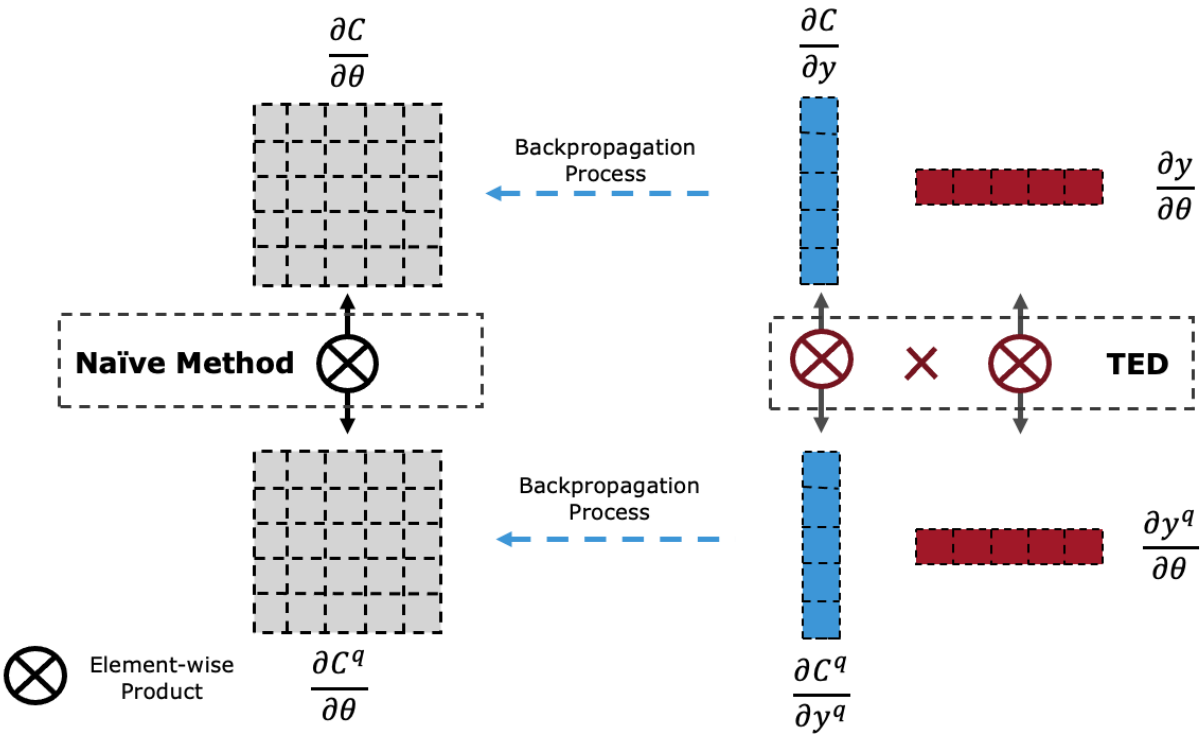



Figure 22: Comparison between naive method and MetaStore.

Therefore, Eq. 62 holds. □

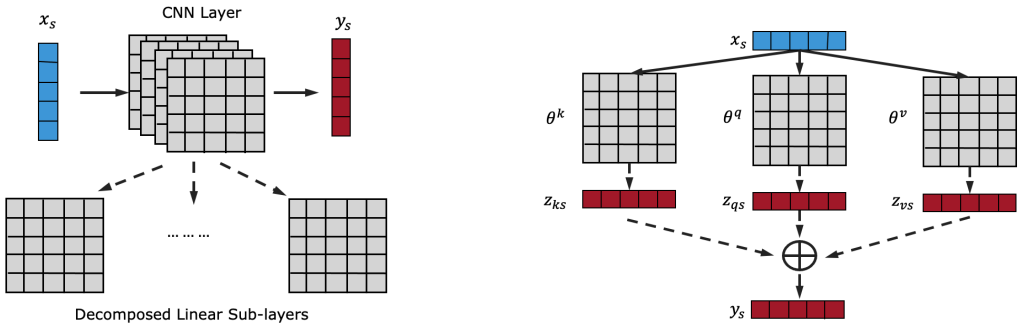
**Time Complexity.** As discussed in Sec. 16.1, a prefix gradient has $D^{out}$ dimensions, while a suffix gradient has $D^{in}$ dimensions. Therefore, the time complexity of MetaStore is $O(D^{in}+D^{out})$. Because the size of the original gradients is $D^{out} \times D^{in}$, the time complexity of computing the inner product directly on the gradients is $O(D^{out} \times D^{in})$. Theoretically, MetaStore speeds up the P2P operation by $O(\frac{D^{out} \times D^{in}}{D^{in}+D^{out}})$. Given a linear layer in VGG with $4096 \times 4096$ parameters, the speed up can be up to 2048 fold.

## 17.2 P2P Operator: Convolutional Layers

As discussed in Sec. 16.2, a convolutional layer can be decomposed into a set of linear layers. Given a convolutional layer whose parameters form a tensor $\theta$ with a shape of $(C^{in}, C^{out}, K)$, because the $K$ is often very small (e.g., K=9 in the VGG16 model), we could decompose $\theta$ into $K$ linear sub-layers $\theta^i$.

Therefore, when computing the inner product of two gradients produced in a convolutional layer, intuitively we could leverage the P2P operator designed for the linear layers to compute the inner product between the gradients with respect to $\theta^i$ and then sum up all the results.

Let's denote two data samples as $x_1$ and $x_2$ and the corresponding outputs of a convolutional layer $\theta$ as $y_1$ and $y_2$. We denote their loss values as $C_1$ and $C_2$ and thus the




(a) **Convolutional Layer** (b) **Self-attention Layer**

Figure 23: Extension to convolutional and self-attention layers: (a) For convolutional layer, MetaStore decomposes it into a series of linear layers; (b) For self-attention layer, MetaStore handles each of its linear sub-layers separately.

gradients as $\nabla_\theta C_1$ and $\nabla_\theta C_2$. Lemma 6 shows how to use the prefix and suffix gradients to directly compute $\nabla_\theta C_1 \cdot \nabla_\theta C_2$ in a convolutional layer.

**Lemma 6.** $\nabla_{\theta^k} C_1 \cdot \nabla_{\theta^k} C_2 = \sum_s^{S-K} \sum_{\tilde{s}}^{S-K} [\frac{dC_1}{dy_1^{s-K}} \cdot \frac{dC_2}{dy_2^{\tilde{s}-K}}] \cdot [x_1^s \cdot x_2^{\tilde{s}}]$

*Proof.* From Lemma 59, we have,

$$\begin{aligned}
\nabla_{\theta^k} C_1 \cdot \nabla_{\theta^k} C_2 &= [\sum_s^{S-K} \frac{dC_1}{dy_1^{s-K}} \cdot x_1^s] \cdot [\sum_{\tilde{s}}^{S-K} \frac{dC_2}{dy_2^{\tilde{s}-K}} \cdot x_2^{\tilde{s}}] \\
&= \sum_s^{S-K} \sum_{\tilde{s}}^{S-K} [\frac{dC_1}{dy_1^{s-K}} \cdot x_1^s] \cdot [\frac{dC_2}{dy_2^{\tilde{s}-K}} \cdot x_2^{\tilde{s}}] \\
&= \sum_s^{S-K} \sum_{\tilde{s}}^{S-K} [\frac{dC_1}{dy_1^{s-K}} \cdot \frac{dC_2}{dy_2^{\tilde{s}-K}}] \cdot [x_1^s \cdot x_2^{\tilde{s}}]
\end{aligned} \tag{64}$$

□

By Lemma 6 MetaStore could use Eq. 64 to compute $\nabla_{\theta^k} C_1 \cdot \nabla_{\theta^k} C_2$.

**Time Complexity.** By Eq. 64 the time complexity of computing $\nabla_{\theta^k} C_1 \cdot \nabla_{\theta^k} C_2$ is $(S-K)^2 \times (C_{out} + C_{in})$. Because $\nabla_\theta C_1 \cdot \nabla_\theta C_2 = \sum_k \nabla_{\theta^k} C_1 \cdot \nabla_{\theta^k} C_2$, the total time complexity of MetaStore calculating the gradient inner product on a CNN layer is $K \times (S-K)^2 \times (C_{in} + C_{out})$.

The time complexity of directly using the original gradients to compute the inner product would be $K \times C_{in} \times C_{out}$ – which is identical to the number of the parameters. The potential speedup is thus $\frac{C_{in} \times C_{out}}{(S-K)^2 \times (C_{in} + C_{out})}$. Therefore, whether MetaStore will win or not depends on the number of features ($S$) of the input samples and the number of input and output channels ($C_{in}$ and $C_{out}$). For most of the popular models, $S$ decreases with the number of layers due to the convolution operation, while $C_{in}$ and $C_{out}$ increase.

Therefore, the number of parameters in the later convolutional layers of a DNN model is often much larger than its early layers.

For example, the first convolutional layer of the VGG16 model only contains $9 \times 3 \times 64$ parameters, where $9$ is the kernel size, while its last layer contains $9 \times 512 \times 512$ parameters. When training a VGG16 model on CIFAR-10 dataset, the number of input features in each channel is $32 \times 32$ in the first convolutional layer, while it is only $1 \times 1$ in the last CNN layer. Thus, MetaStore tends to significantly outperform the naive method on the later convolutional layers, while it can be slower on the early layers.

In practice, MetaStore could leverage the above time complexity as a cost model to select a low cost execution strategy.

## 17.3 P2P Operator: Self-Attention Layers

As discussed in Sec. 16.3, a self-attention layer is formed by three linear sub-layer $\theta^k$, $\theta^q$ and $\theta^v$. Therefore, MetaStore can directly leverage the strategy designed for the linear layer to compute the inner product between the gradients with respect to each sub-layer and then at the end multiply the results.

**Time Complexity.** Let's denote two data samples as $x_1$ and $x_2$, the corresponding output of a CNN layer $\theta$ is $y_1$ and $y_2$. The time complexity of MetaStore is $S^2 \times (H + H)$, where $S$ is the length of the input sequence and $H$ represents the number of the dimensions of the hidden vector for each word. The time complexity of the naive solution that pre-computes and stores the original gradients is $H \times H$. So the potential speedup of MetaStore is $\frac{H}{2 \times S^2}$.

In a standard BERT model, $H = 768$. Therefore, as long as the length of each sequence $S$ is smaller than $\sqrt{384}$, MetaStore will win. We find this often holds on most of the popular benchmark NLP datasets [130].

# 18 Meta-data Analytics: Batch Operators

Next, we discuss our strategy to efficiently support the P2B operator in Sec. 18.1. Then in Sec. 18.2 we show how to leverage the efficient P2B execution strategy to support B2P and B2B operators.

## 18.1 P2B Operator: No Gradient Restore

The point-to-batch (P2B) operator estimates the contribution of one training sample $x_i$ on the prediction results of a batch of testing samples $B_j$. By the concept of the meta-gradient introduced in Sec. 14.1, MetaStore measures this as the average inner product similarity between the gradients of the training sample and any testing sample in the batch. We denote this as $\mathcal{I}^{avg}(x, B)$.

By Sec. 16, MetaStore has already preprocessed and stored the gradient of each training sample beforehand as a $< prefix, suffix >$ pair. Thus, the pair can be directly fetched. However, MetaStore has to obtain on the fly the gradient of the testing samples unseen before using model replay, as described in Sec. 15. Given a batch of testing samples, MetaStore can get their gradients in two ways: (1) for each sample, we get its gradient in the format of $< prefix, suffix >$ pair as described in Sec.15; or, (2) we directly get the average gradient of this batch. The existing deep learning infrastructures such as Pytorch readily provide this interface, because deep learning typically updates the model parameters based on the average gradient of a batch of training samples.

The advantage of the first approach is that MetaStore can directly call the efficient P2P operators introduced in Sec. 17 to compute $\mathcal{I}^{avg}(x, B)$. However, it has to iteratively compute the inner product for each pair of training and testing samples. When the testing batch is large, this will become expensive.

On the other hand, $\mathcal{I}^{avg}(x, B)$ is equivalent to the inner product between the gradient of $x_i$ and the *average gradient* of the testing batch. Therefore, if MetaStore restores the full gradient of the training sample from the $< prefix, suffix >$ pair and extracts the average gradient of the testing batch using the second approach, then it would be able to compute $\mathcal{I}^{avg}(x, B)$ with *one single* inner product operation. However, restoring the training sample from the $< prefix, suffix >$ pair tends to be expensive – often much more expensive than the inner product itself.

We design an efficient P2P execution strategy which uses *one single* inner product operation to compute $\mathcal{I}^{avg}(x, B)$, while *not restoring* the full gradient of the training sample. This strategy is built on Lemma 7.

**Lemma 7.** Let $\nabla_\theta C$ denote the gradient of training sample $x$ and $\bar{G}^t$ the average gradient of the testing batch, given a linear layer with the shape of $(D^{in}, D^{out})$, Eq 65 holds.

$$\mathcal{I}^{avg}(x, B) = \nabla_\theta C \cdot \bar{G}^t = x^T \cdot \bar{G}^t \cdot \frac{dC}{dy} \tag{65}$$

*Proof.*

$$\mathcal{I}^{avg}(x, B) = \nabla_\theta C \cdot \bar{G}^t \ = \sum_{i=0}^{D^{in}} \sum_{j=0}^{D^{out}} [(\nabla_\theta C)_{(i,j)} \cdot \bar{G}^t_{(i,j)}] \tag{66}$$

From Eq. 54, we have $(\nabla_\theta C)_{i,j} = \boldsymbol{x}_i \cdot (\frac{dC}{dy})_j$. Then:

$$\begin{aligned} \mathcal{I}^{avg}(x, B) &= \sum_{i=0}^{D^{in}} \sum_{j=0}^{D^{out}} [x_i \cdot \bar{G}^t_{(i,j)} \cdot (\frac{dC}{dy})_j] = x \cdot \sum_{j=0}^{D^{out}} [\bar{G}^t_{(i,j)} \cdot (\frac{dC}{dy})_j] \\ &= x^T \cdot \sum_{j=0}^{D^{out}} [\bar{G}^t_{(i,j)} \cdot (\frac{dC}{dy})_j] = x^T \cdot \bar{G}^t \cdot \frac{dC}{dy} \end{aligned} \tag{67}$$

where $\boldsymbol{x}$ and $\frac{dC}{dy}$ correspond to the ¡prefix, suffix¿ pair of the training sample. This concludes the proof of Lemma 7. □

Eq. 67 shows that we could directly execute the P2B operator using the prefix gradient $\frac{dC}{dy}$ and the suffix gradient $x$.

**Time Complexity.** The time complexity is $D^{in} \times D^{out}$, which does not rely on the size of the batch. Therefore, this strategy scales to large testing batches.

Extending this method to the convolutional and self-attention layers is straightforward. MetaStore decomposes their parameters into a set of linear sub-layers. Using the average gradient of the testing batch w.r.t. each linear sub-layer and the pre-stored ¡prefix,suffix¿ pair, it first calculates the partial inner product as described above and then aggregates up the partial results.

## 18.2 Other Operators: B2P and B2B

Unlike the P2P and P2B operators, the B2P and B2B operators involve a batch of training samples which have their prefix and suffix gradients already maintained in MetaStore storage. An intuitive method to execute these two types of operators would thus be to first restore the original gradient from the prefix and suffix gradients for each training sample, compute the average gradient for this batch, then use model replay to extract the gradient or the average gradient for the testing samples, and finally compute the inner product. This method only needs to compute the inner product once. However, as we have discussed in Sec. 18.1 and confirmed in the experiments (Sec. 19.4), restoring the original gradient from the prefix and suffix gradient typically is even more expensive than the inner product operation itself, thus not acceptable.

After ruling out restoring the gradients as an option, the only way left is to iterate over each training sample in the batch, then to call the P2P or P2B operator to compute the inner product, and lastly, to take the average.

More specifically, for the B2P operator, MetaStore could extract the gradient in the format of ¡prefix, suffix¿ pair for the single testing sample via model replay and call the P2P operator. Another option would be to extract the original full gradient and call the P2B operator instead using our strategy presented in Sec. 18.1. For the B2B operator, directly calling the P2B operators is the obvious choice. Because users tend to form a batch of training samples mimic the DNN training process, the size of the training batch usually is set according to the *batch size* hyper-parameter (typically 64 or 128). Thus, the cost of iterating over each training sample in a batch is acceptable.

# 19 Experiments

Our experimental study focuses on the following questions:
• Storage: Does MetaStore reduce the storage footprint of gradient meta-data and thereby offer practical feasibility?
• Execution Time: To what degree does MetaStore speed up the execution of gradient-based analytics queries compared to unoptimized methods?
• Preprocessing: How efficiently can we collect the meta-data in MetaStore?
• How useful are our analytics interfaces in applications?

## 19.1 Experiment Setup

**Settings.** All the experiments are implemented in Python3.7 on Pytorch. We conduct all experiments on a virtual cloud instance with Intel Xeron G6248 CPU, 0.5 TB Memory, a SSD storage disk with 2TB space, and one V100 GPU with 32G memory.
**Datasets.** We evaluate our method with two benchmark datasets, namely, **CIFAR10** image and **AGNews** [126] text dataset. CIFAR10 dataset contains 50,000 images from 10 classes. Each image has the dimension of $3 \times 32 \times 32$. AGNews dataset contains 30,000 sentences from four classes, where each sentence contains 6 to 89 words.
**Baseline Methods.** In the experiments we only measure the efficiency of the point-to-point (P2P) operators and the point-to-batch (P2B) operators, because the B2P and B2B operators simply leverage the P2P and P2B operators, as discussed in Sec. 18.2.

For the P2P operators, we compare MetaStore with the following baseline methods:

1) **Pre-compute**: We pre-compute the full gradient for all training samples and store them in disk. Once an analytics query is submitted, we retrieve the gradient of the indicated training sample from the disk into GPU memory, extract the gradient for the indicated testing sample in the ¡prefix,suffix¿ pair format, and run the corresponding analytics operators.

2) **Re-compute**: After an analytics query is submitted, it computes the gradient of the training sample on the fly through model replay using the model maintained in GPU.

For the point-to-batch (P2B) operators, we evaluate the *Iterate* and *Reconstruction* methods discussed in Sec. 18. Both methods leverage our compact ¡prefix, suffix¿ storage structure to reduce the I/O costs when collecting the gradients of training samples.

1) **Iterate**: This method extracts the gradients for each indicated testing sample in the ¡prefix,suffix¿ pair format as described in Sec.15, and then calls our *optimized* P2P operator to compute the inner product between the training samples and each testing sample in the query batch and then compute the average.

2) **Reconstruction:** This method extracts the average gradient for the testing batch through model replay, and then reconstructs the gradients of training samples from the ¡prefix, suffix¿ pair. Finally, it directly calculates the similarity between the gradient of a training sample and the average gradient of the testing batch. Therefore, *Reconstruction* only computes the inner product once.

Unlike the P2P operators experiments, in the P2B experiments we don't compare against *Pre-compute* and *Re-compute* baselines, because *Iterate* and *Reconstruction* leverage our compact ¡prefix,suffix¿ storage structure and optimized P2P operator. Thus, they are clearly more efficient than *Pre-compute* and *Re-compute*. Similarly, although *Reconstruction* could work for P2B operator, we don't compare against it in the P2P experiments. This is because for the P2P operator, *Reconstruction* does not reduce the number of inner product computations, while introducing extra computation costs to reconstruct the original gradient from the ¡prefix, suffix¿ structure. It is thus guaranteed to be worse than *Pre-compute*.

**Deep Neural Network Architectural Models.** We evaluate our method on two popular deep neural network architectures, namely VGG16 [93] and BERT [26]. As a well known model for computer vision tasks, VGG16 consists of 13 Convolutional layers and three linear layers. The BERT Model, popular in natural language processing, contains 12 Attention layers and one linear layer. We trained a VGG16 model on CIFAR10 dataset and a BERT model on the AGNews dataset through finetuning. We trained each model for 5 epochs and saved the model checkpoints.

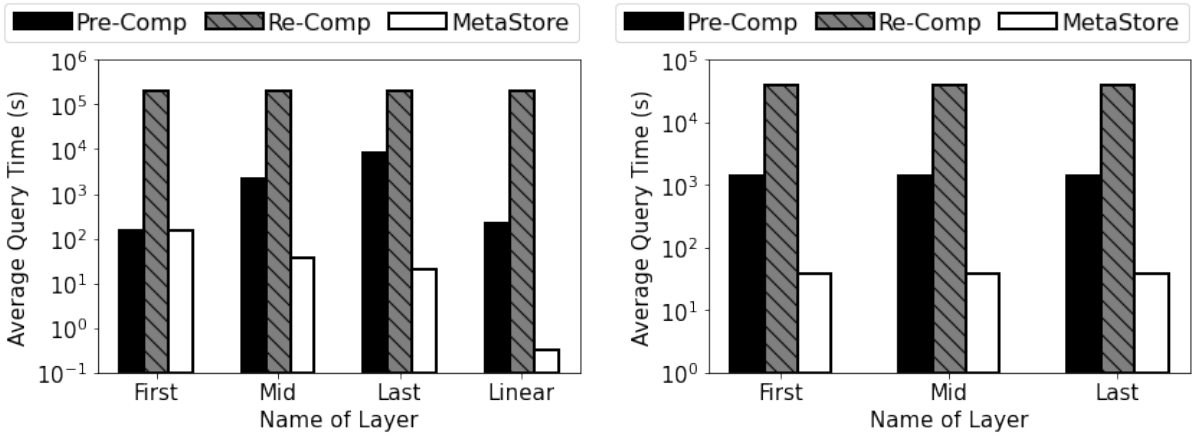
| Storage Cost (MB) | | | | |
|---|---|---|---|---|
| Layers | Shape | MetaStore | Full Gradient | Disk Space Saving |
| VGG16-Conv1 | $9 \times 3 \times 64$ | 2744 | 69 | $0.025\times$ |
| VGG16-Conv7 | $9 \times 128 \times 256$ | 1310 | 23593 | $18.0\times$ |
| VGG16-Conv13 | $9 \times 512 \times 512$ | 163 | 94371 | $578\times$ |
| VGG16-Linear1 | $512 \times 10$ | 21 | 205 | $9.76\times$ |
| BERT-SAL1 | $3 \times 768 \times 768$ | 2949 | 70779 | $24.00\times$ |
| BERT-SAL6 | $3 \times 768 \times 768$ | 2949 | 70779 | $24.00\times$ |
| BERT-SAL11 | $3 \times 768 \times 768$ | 2949 | 70779 | $24.00\times$ |
| BERT-Linear1 | $768 \times 4$ | 31 | 122 | $3.93\times$ |

Table 1: The storage cost: MetaStore VS **Full Gradient**.

## 19.2 Storage Costs

In this set of experiments, we evaluate the storage costs and savings of the MetaStore's prefix/suffix gradient strategy of storing decomposed gradients. We evaluate the storage cost for 10,000 training samples randomly sampled from the training set, because the baseline cannot handle the whole training set. Following previous work [43], for the VGG16 model, we report the storage costs of the first, mid and last convolutional layers and the linear layer, while for the BERT model, we report the storage costs of the first, mid and last self-attention layer and the last linear layer.

Table 1 shows these storage costs. We see that compared to storing the original gradients, MetaStore reduces the storage costs by up to 578x for the VGG16 model solution.




(a) **VGG16-CIFAR10** (b) **BERT-AGNews**

Figure 24: The End-to-End Query Execution Time of P2P Operator for Different Layers in VGG16 and BERT Model.

The only exception is the first convolutional layer that featuring only a few parameters, has a small gradient and thus not much disk space is saved in this case. As an optimization, MetaStore leverages the space complexities discussed in Sec. 16 as cost model to determine when to store the full gradient versus the gradients in ¡prefix,suffix¿ pair format.

Similarly, for the BERT model, MetaStore reduces the storage costs by 24x. However, for this model, both methods need more disk space than the VGG16 model, because the BERT-AGNews model contains many more parameters than the VGG16-CIFAR10 model. Also, each layer in the BERT-AGNews model generates a larger number of input and output features for each training sample compared to the VGG16-CIFAR10 model.

## 19.3 P2P Operator: End-to-End Execution Time

In this experiment, we evaluate the end-to-end execution time of the P2P operator which computes the inner product between the gradients of two data samples. This execution time includes the times for calculating the gradients of the testing samples by model replay, loading the gradients of the training samples into GPU memory, and running the corresponding analytics operators.

### 19.3.1 Execution Times for Different DNN Layers

First, we compare MetaStore against the *Pre-compute* and *Re-compute* methods (Sec. 19.1) on the VGG16-CIFAR10 and BERT-AGNews models. Similar as with the above storage experiments, we evaluate the first, middle and the last convolutional layer of the VGG16 model and the first, middle and the last self-attention layer of the BERT Model. We ran-

domly select one testing sample from the testing set. For each pair of training sample and this chosen testing sample, we run the P2P operator. We use 10,000 training samples and thus call the P2P operator 10,000 times. We repeat the experiment 10 times and report the average execution time.

Fig. 24(a) (in **log scale**) shows that for the VGG16-CIFAR10 model, MetaStore is up to 1,000 times faster than *Pre-compute*, and 7 orders of magnitude faster than *Re-compute*. In particular, *Pre-compute* is slower on the later convolutional layers, while MetaStore improves speed there. This is because that the complexity of *Pre-compute* increases linearly with the number of parameters, and indeed the later convolutional layers have more parameters. On the other hand, the complexity of MetaStore increases linearly with the size of the input features, while the size of the input feature is smaller in the later layers compared to the early layers. This is common for CNN networks, since the convolution operation naturally shrinks the size of the features. Further, the execution time of *Re-compute* does not vary much for different layers. This is because the performance bottleneck of this method is performing forward and backward propagation on the entire network to produce the gradients.

For the BERT model, MetaStore is about 10 to 100 times faster than Pre-compute and 100 to 1000 times faster than Re-compute. Because different self-attention layers in BERT have the same architecture, their performance does not vary much across different layers. The speed up of MetaStore for BERT is smaller than for the VGG16 model. This is because on the BERT model MetaStore has a relatively speaking smaller advantage on storage costs, as discussed in Sec. 19.2.

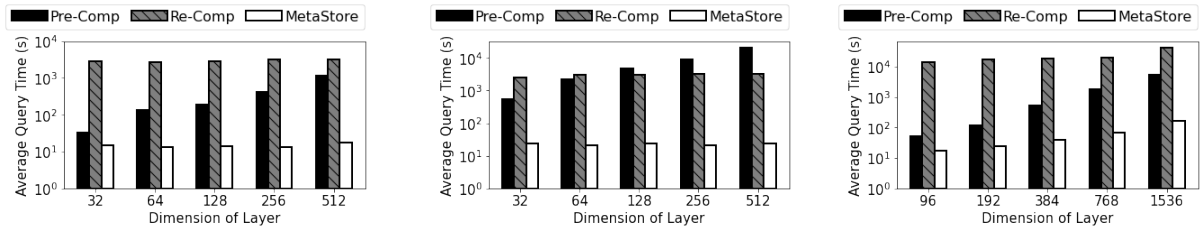
### 19.3.2 Varying Number of Dimensions of Layers

In this set of experiments, we evaluate MetaStore's performance on DNN layers with a varying number of dimensions. To achieve this, for the linear layer, we append one additional linear layer before the last layer in VGG16. Similarly, for the convolutional layer, we append one additional convolutional layer after the last convolutional layer in VGG16. We refer to these two "extended" models as VGG16-Linear and VGG16-Conv, respectively. We then vary the number of dimensions of these new layers. For the self-attention layer, we directly vary the input and output dimension of each self-attention layers. We name this model BERT-Att.

For the VGG16-Linear model, to ensure the appended layer is aligned with the previous layers, we keep the input dimensions fixed and only vary the output dimensions from 32 to 512. Similarly, for the VGG16-Conv model, we fix the number of input channels and vary the output channels from 32 to 512. Thereafter, we focus on comparing the end-to-end execution time on the new layer of each model. For the BERT-Attention model, we vary the input and output dimensions of each Self-Attention layers from 96 to 768. We report the execution time of the last self-attention layer.

As depicted in Fig. 25, MetaStore is up to 1000 times faster than both baseline methods

in all experiments. For the VGG16-Conv and the VGG16-Linear models, as shown in Fig. 25(b) and Fig. 25(a), respectively, we observe that for all three types of layers, the execution time of the Pre-compute method increases quickly as the output dimensions get larger, while the query time of MetaStore does not increase significantly. This can be explained by the time complexity of *Pre-compute* which equals the input dimensions multiplied by the output dimensions, while the time complexity of MetaStore equals the input dimensions plus the output dimensions, as discussed in Sec. 16. For the BERT-Attention model, MetaStore is up to 1,000 times faster than both baseline methods. In all experiments, the execution time of *Re-compute* method does not change significantly and is much slower than the other two methods in most cases. This is because calculating the gradient of a single layer on the fly is expensive, which dominants the execution time in this case.




(a) **VGG16-Linear** (b) **VGG16-Conv** (c) **BERT-Att**

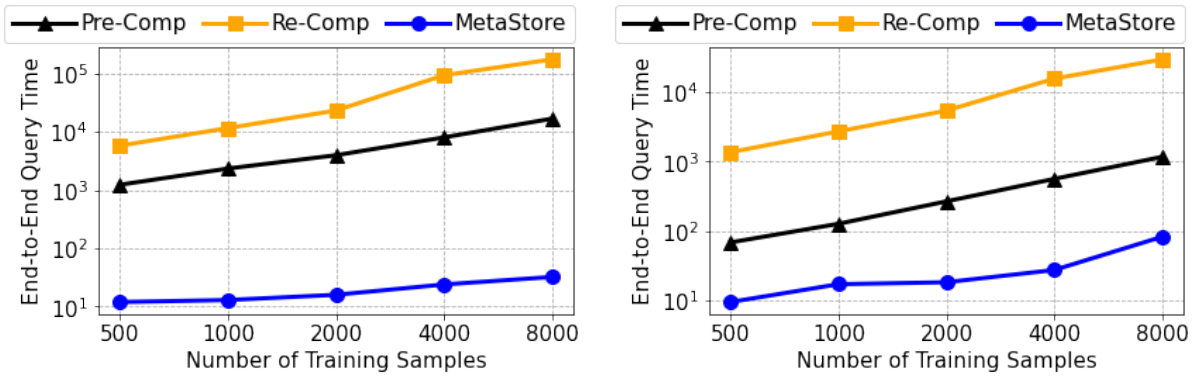
Figure 25: The End-to-End Query Execution Time of P2P Operator: Varying the Dimension of Different Type of Layers.

### 19.3.3 Vary the Number of Training Samples

We vary the number of training samples for each query from 500 to 8,000 and compare MetaStore against the Pre-Compute and Re-Compute methods. We measure the cumulative total time of running 100 queries on the last convolutional and the last linear layer in the VGG16-CIFAR10 model and the last self-attention layer in the BERT-AGNews model. We cache the gradients in the memory, when possible, using LRU as cache replacement policy. As shown in Fig. 26, the cumulative time of MetaStore increases much slower than for the baseline methods. Specifically, MetaStore only gets about 5 times slower when increasing the number of training samples from 500 to 8000 on the VGG16-CIFAR10 and BERT-AGNews models, while the execution time of *Pre-Compute* and *Re-Compute* increase 12-15 times in both cases. This is because MetaStore can cache more data samples in memory due to its efficient storage strategy, therefore significantly reducing the I/O costs.

## 19.4 P2B Operator: Execution Time

We evaluate the performance of our optimized method (Sec. 18.1) for the P2B operator. In this experiment, we compare *Iterate* and *Reconstruction* as baseline methods. The recon-

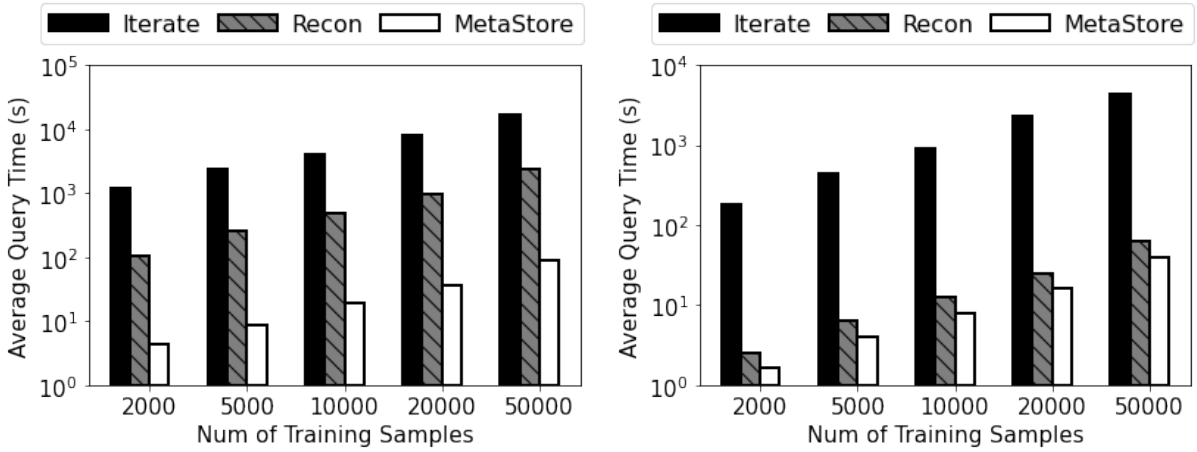



(a) **VGG16-CIFAR10** (b) **BERT-AGNews**

Figure 26: The End-to-End Query Execution Time of P2P Operator: Varying the Number of Training Samples.

struction method leverages our prefix/suffix gradients insights, thus significantly reducing its I/O costs.

As shown in Fig. 27, our method is at least 2 times faster than the baseline methods in all experiments. Compared with the reconstruction method, our method speeds up the execution by up to 10x, because it directly computes the results on the ¡prefix, suffix¿ pairs of training samples, and thus avoids reconstructing large gradients for the training samples.

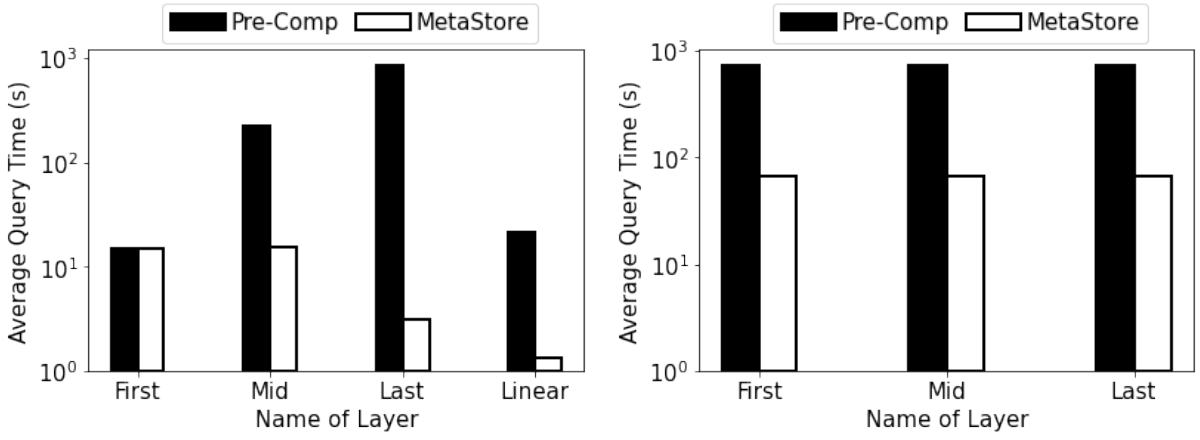



(a) **VGG16-CIFAR10** (b) **BERT-AGNews**

Figure 27: The End-to-End Query Execution Time of P2B Operator: Varying the Number of Training Samples.

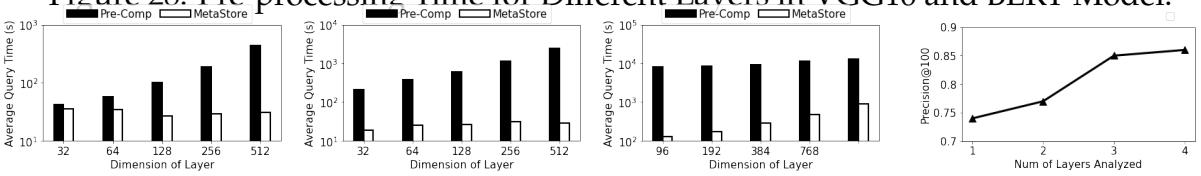
## 19.5 Meta-data Collection and Storage Times

We evaluate the time of extracting and storing the gradient of 10,000 training samples. We compare MetaStore against computing and storing the full gradients. Again, we measure




(a) **VGG-16-CIFAR10** (b) **BERT-AGNews**

Figure 28: Pre-processing Time for Different Layers in VGG16 and BERT Model.




(a) **VGG16-Linear** (b) **VGG16-Conv** (c) **BERT-Attention** (d) **Detect Noisy Labels**

Figure 29: Fig.(a)-(c):The Meta-data Collection Time: Varying the # of Dimensions of Different Types of Layers. Fig.(d): The precision of detecting mislabeled training samples using MetaStore.

the collection time on the first, mid, and last convolutional layers and the linear layer in the VGG16-CIFAR10 model and the first, mid, and last self-attention layers and the last linear layer in the BERT-AGNews model. Fig. 28 shows that MetaStore is up to 1,000 times faster than the baseline. This is because although both methods use the same forward and backward propagation process to extract meta-data, MetaStore only needs to log the small prefix and suffix matrices into the storage.

Similar to the trend in the storage cost experiments, the baseline takes more time to collect meta-data on the later convolutional layer in the VGG16 model in comparison to MetaStore. Again, this is because the later convolutional layers in the VGG16 model have more parameters than the earlier convolutional layers.

We also evaluate the meta-data collection time by varying the number of dimensions of the target layers. As shown in Fig. 29, MetaStore consistently outperforms the baseline. As the number of dimensions increases, the collection time of the baseline increases linearly, while MetaStore only becomes slightly slower. This is because the size of gradient equals the input dimension cardinality multiplied by the output dimension cardinality, and the size of prefix and suffix matrix equals the input dimension cardinality plus the output dimension cardinality. The latter is much smaller.

## 19.6 The Usefulness of Gradient-based Analytics

Finally, we use data debugging as an example to showcase the gradient-based analytics enabled by MetaStore is indeed useful.

As discussed in Sec. 14.2, users can use the P2B operator to discover mislabeled objects (data debugging). That is, the users first specify a batch of testing samples, invoke the *P2B* operator on each training sample, and then rank the training samples based on the gradient inner product produced by the P2B operator. The $k$ training samples (where $k$ is a user defined input parameter) with the smallest values are the least influential samples, and therefore the most likely to be mislabeled. MetaStore also allows the users to specify the layers of the DNN model to be involved in the analysis.

In this study, we train a VGG16 model using the CIFAR10 dataset. We inject 1% (500) mislabeled samples into the training set by randomly flipping their original labels. We randomly select 100 testing samples to form the batch. We gradually add the layers of the DNN model, starting with only the last linear layer and then adding the last, middle, and first convolutional layers step by step. Fig. 29(d) shows the precision of the 500 ($k$ = 500) mis-labeled objects that MetaStore identifies. The precision increases from 0.75 to 0.85 as more DNN layers are considered.

# 20 Related Works

**DNN Diagnosis Tools.** A plethora of developed tools have been researched for diagnosing DNN models, some of which involve meta-data. Among them, MISTIQUE [105, 104] supports compactly storing the meta-data, namely, feature embeddings and losses using data compression techniques like quantization. DeepEverest [43] speeds up the model diagnosis queries on storage. However, none of them support gradient-based diagnostic queries due to the lack of techniques to compactly store and efficiently analyze gradients – possibly due to their size.

To speed up DNN training, some works [38, 78, 23] proposed techniques to quickly compute the gradients. Although these techniques could be used by our MetaStore to collect gradients, they do not solve the problem of efficiently storing and analyzing gradients. Lastly, several works [6, 51, 51, 118, 87, 92, 68] target the visualization of meta-data, which could potentially be used for human-driven model examination. However, none of them use gradients.

**Robust Deep Learning.** Researchers have used meta-data in the training process to make DNN models robust to noisy data and adversarial attack. For example, in [89, 15, 98, 74, 104], the authors use training losses and feature embeddings to detect erroneous training data samples. Some works [77, 114, 127] overcome the problem caused by mislabeled training samples by weighting them according to the inner product between the gradients of training data and the gradients of validation data. Others [60, 80] use the gradients of data samples as a metric to evaluate the confidence of model prediction and improve model robustness to adversarial attacks.

In contrast, people have used gradients as a signal to perform adversarial attacks on DNN models [79, 120]. Some methods [15, 1] leverage statistics of gradients to identify potential data leakage of DNN models. Some other methods use the gradients to modify the training process and search for hyper-parameters [115, 64, 13, 50] to improve DNN models' performance. All of these prior works potentially could benefit from our solution to speed up their training and this way scale to large datasets.

**Gradient Compression.** Federated learning may need to transfer gradients from the clients to the servers. To reduce the communication costs, researchers have proposed techniques [62, 17, 5, 111, 12, 9] to compress the gradients by approximation. Some works [5, 111, 3, 9, 12] use quantization and sparsification techniques to compress the gradients by preserving the large gradient values while discarding the small ones. Some other works [55, 108, 107] use matrix factorization to decompose big gradients. But none of them are able to recover the exact gradients.

Our MetaStore instead re-uses the intermediate results produced in the back propagation process to compactly store and efficiently analyze the gradients, without conducting extraneous operations such as compression and matrix factorization. Thus, they are overhead free. MetaStore could *exactly* restore a gradient from this compact storage, rather than an approximation. Therefore, our techniques could be of benefit to federated learning research.

# Part IV
# Conclusion and Future Directions

## 21 Conlusions

### 21.1 ELITE: Outlier Removal From Training Data

In this work, we propose ELITE that addresses a fundamental problem in semi-supervised and unsupervised deep anomaly detection, namely requiring a clean training data not polluted by anomalies. LANCET solves above problems by proposing a novel optimization methodology. Unlike the classical semi-supervised classification methodology, ELITE uses labeled examples as validation set and continuously discovers the anomalies in the polluted training data and learns a better deep anomaly model based on the cleaned training data. Our experiments in rich variety of scenarios confirm ELITE's superiority to the state-of-the-art and its robustness to polluted training data.

### 21.2 Lancet: Labeling Complex Data At Scale

In this work, we tackle the challenging problem of how to produce a sufficient number of labels for data sets void of labels so to train label thirsty machine learning models with minimal manual labeling effort.

Our proposed solution, LANCET, solves three critical interdependent subproblems essential for an effective labeling solution, namely, what objects to label, how to automatically produce labels, and when to terminate labeling. LANCET does so in a principled way based on a solid theoretical foundation. Our experiments using multiple public data sets demonstrate that LANCET significantly outperforms all alternative solutions in both accuracy of labels generated and quality of the machine learning models, including weak supervision and active learning based methods.

### 21.3 MetaStore: Meta Data Analytics for Training Data Curation

In this work, we propose MetaStore to efficiently collect, store, and analyze meta-data produced in the DNN training process. The key techniques of MetaStore address the challenges caused by the size of the gradients and thus for the first time enable the gradient-based analytics for data debugging and model interpretation. The key idea is that although in many DNN layers a gradient tends to be huge, its prefix and suffix gradients produced in the backpropagation process are much more compact to store. From these prefix and suffix gradients, collected almost for free, MetaStore succeeds to exactly restore the original gradients. Moreover, MetaStore implements a rich set of operators that support various gradient-based analytics tasks directly on top of compact gradient

storage structures. Thus they are much more efficient than working on the original huge gradients. Our experiments show that MetaStore significantly reduces storage costs and query execution times by orders of magnitude compared to baseline solutions.

# 22 Future Directions

## 22.1 ELITE: Outlier Removal From Training Data

In the ELITE project, we propose a robust deep outlier detection method. The method only requires a very small set of labeled data samples to significantly improve the robust deep outlier detection models. More specifically, ELITE utilizes the labeled data samples as the validation data instead of the training data. It iteratively identifies the outliers in the raw dataset according to their influence on the validation loss. ELITE leverages the meta-gradient to estimate the influence of each training sample on the validation loss.

However, the current system assumes the labeled data sample are randomly sampled from the unlabeled dataset and contains various type of outliers. However, it is often infeasible for the users to collect a rich set of outliers due to the rareness and variety of outliers. Thus, it is natural to develop a human-in-the-loop deep outlier detection method with meta-gradient as the next step. We need to solve two critical challenges to develop such a method.

First, we should develop a novel algorithm that selects the data samples for users to label. This problem is commonly addressed by the active learning problem. However, the active learning algorithms are commonly designed for traditional supervised learning scenarios where the labeled samples are used as the training data. As a novel training scheme, the meta-gradient utilizes the labeled samples as the validation samples. Designing the corresponding active learning principles for meta-gradient based methods is not only critical for the ELITE project but also serves as the foundation of other meta-gradient based algorithms.

Second, the current ELITE methods need approximately three times computing time compared with the standard supervised training scheme. The extra cost may be acceptable in a one-shot learning scenario. However, iteratively training the model under such a high computing cost could be infeasible. As a human-in-the-loop method, the newly developed method should be able to adjust the model based on human feedback quickly. Thus, exploring additional methods to accelerate ELITE's training process is essential. An important opportunity lies under the historical model predictions during ELITE's training process. The current ELITE method iteratively calculates the meta-gradients of each training sample and detects outliers in each training epoch. However, this is unnecessary. A large set of training samples are always determined as inliers or outliers during the entire model training process. The meta-gradient calculation of the corresponding training samples could be avoided if we carefully manage and leverage the signals during the model training process.

## 22.2 Lancet: Labeling Complex Data At Scale

In the LANCET project, we tackle the challenging problem of producing a sufficient number of labels for data sets for label-thirsty machine learning models, such as Deep Neural Networks. Our proposed solution solves three critical interdependent subproblems essential for an effective labeling solution in a principled way based on a solid theoretical foundation. The current LANCET assumes there is a domain expert which could accurately annotate data samples. LANCET also incorporates semi-supervised distribution learning models, which could learn the intrinsic distribution of the dataset and accurately propagate labels from labeled samples to unlabeled samples. As a future direction, it is worth relaxing these assumptions and requirements to make LANCET more suitable for real-world applications.

In many real applications, it is hard to obtain accurate labels. For example, training machine learning models in recommendation systems often rely on the labels collected by implicit user feedback, such as ads clicks. As another example, in medical applications, the data samples sometimes are annotated by users with different levels of medical expertise, such as experienced doctors and new grad students from medical colleges. The noisy labeling oracles that annotate data samples with inaccurate labels may cause new problems in the current system.

First, most semi-supervised distribution learning approaches assume that all the labeled samples are correct. The mislabeled data samples potentially degrade the quality of learned distribution. Consequently, the similar data samples in the learned feature embedding may not belong to the same class, and it is not safe to propagate the labels. To address this, a robust semi-supervised distribution learning algorithm is required.

Second, even if a high-quality distribution is obtained, propagating labels from labeled samples to unlabeled samples is also dangerous. For example, when a group of unlabeled samples is discovered. To avoid unnecessary labeling costs, the current system selects a representative data sample from this group and sends it to the oracles for labeling. Then the label of the selected sample will be propagated to the entire group of data samples. However, the entire group of data samples will be propagated with incorrect labels when the labeling oracles are noisy. One possible approach to tackle this challenge is to select a few data samples for labeling, instead of a single one, to validate the provided labels and estimate the accurate label. Furthermore, the developed approach should be able to jointly incorporate label oracles with different levels of domain expertise and automatically trade off between the label propagation accuracy and manual labeling cost.

Third, although many active learning algorithms have been proposed to select the most informative data samples for manual labeling, most of them assume the labeling oracles are error-free. It is an exciting direction to develop theories and algorithms for active learning problems with noisy label oracles.

## 22.3 MetaStore: Meta Data Analytics for Training Data Curation

We have established the foundational components of a metadata analytic system, MetaStore. MetaStore currently includes a lightweight metadata collector, a compact metadata storage, and an efficient metadata analytic engine that supports a set of analytic primitives. There are significant opportunities to enable users to fully leverage metadata in real applications. In this section, we will discuss three future directions for the MetaStore project.

### 22.3.1 Cost Based Optimizers

The first direction is to design optimizers to minimize the query execution costs. Running an analytic query with the MetaStore requires the users to define the following variables,

- Type of Meta Data, e.g., loss, feature embeddings, gradients, etc.
- Model Checkpoint, e.g., at the end of the fifth epoch during training.
- Model Layer, e.g., the last linear layer.
- Training Samples, e.g., the training samples collected in the last month.
- Query Samples, e.g., the testing samples that the model made incorrect predictions.

Even though MetaStore already significantly accelerates the query execution process, running the analytic queries can still be time and storage-consuming. Furthermore, instead of the precise analytic result, the users are often only interested in the top $k\%$ of training samples that are most related to the target task. This motivates us to design query optimizers to trade-off between the accuracy of analysis results and query execution efficiency.

### 22.3.2 Human-in-the-loop Analytic

Another future direction is supporting human-in-the-loop analytics to have MetaStore easily incorporate domain-specific requirements. In this way, users can use their domain knowledge to adapt this generic system to better serve their applications. Here arises the question of how the system could leverage the feedback to dynamically adjust the analytics approach so that it can capture the mislabeled samples more accurately in the future. Toward this goal, we need to develop the following components.

First, we need to develop a visualization component for MetaStore, which allows the users to visualize the analytic process. For example, a user should have access to how the data valuation ranking is formed, including which model checkpoint, model layer, and metadata are used in the optimized query executing process. Furthermore, the visualization component should have an interface for users to directly define the query running

heuristics, modify the query optimizers, and provide feedback on the results, such as if a detected sample is mislabeled.

Second, we need a set of algorithms that could automatically utilize the ground truth labels to update the optimizers. For example, users might provide feedback on some detected mislabeled training samples, telling us which are indeed mislabeled and which are not.

# References


[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara. Latent space autoregression for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 481–490, 2019.

[3] A. F. Aji and K. Heafield. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021*, 2017.

[4] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian conference on computer vision*, pages 622–637. Springer, 2018.

[5] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[6] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346, 2015.

[7] J. T. Andrews, E. J. Morton, and L. D. Griffin. Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing*, 6(1):21, 2016.

[8] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3, 2013.

[9] D. Basu, D. Data, C. Karakus, and S. Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] L. Beggel, M. Pfeiffer, and B. Bischl. Robust anomaly detection in images using adversarial autoencoders. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 206–222. Springer, 2019.

[11] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[12] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.

[13] O. Bohdal, Y. Yang, and T. Hospedales. Evograd: Efficient gradient-based meta-learning and hyperparameter optimization. *Advances in Neural Information Processing Systems*, 34:22234–22246, 2021.

[14] L. Cao, Y. Yan, Y. Wang, S. Madden, and E. A. Rundensteiner. Autood: Automatic outlier detection. In *SIGMOD*.

[15] N. Carlini, U. Erlingsson, and N. Papernot. Distribution density, tails, and outliers in machine learning: Metrics and applications. *arXiv preprint arXiv:1910.13427*, 2019.

[16] R. Chalapathy, A. K. Menon, and S. Chawla. Robust, deep and inductive anomaly detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 36–51. Springer, 2017.

[17] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan. Adacomp: Adaptive residual gradient compression for data-parallel distributed training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[18] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 90–98. SIAM, 2017.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[20] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[21] R. D. Cook and S. Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[22] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. R. Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *Advances in neural information processing systems*, pages 6510–6520, 2017.

[23] F. Dangel, F. Kunstner, and P. Hennig. BackPACK: Packing more into backprop. In *International Conference on Learning Representations*, 2020.

[24] N. Das, S. Chaba, R. Wu, S. Gandhi, D. H. Chau, and X. Chu. Goggles: Automatic image labeling with affinity coding. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1717–1732, 2020.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[27] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[28] M. Ducoffe and F. Precioso. Adversarial active learning for deep networks: a margin based approach. *arXiv preprint arXiv:1802.09841*, 2018.

[29] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[30] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[31] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

[32] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189, 2015.

[33] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[34] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[35] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[36] P. Giselsson. Improved fast dual gradient methods for embedded model predictive control. *IFAC Proceedings Volumes*, 47(3):2303–2309, 2014.

[37] I. Golan and R. El-Yaniv. Deep anomaly detection using geometric transformations. In *NeurIPS*, pages 9758–9769, 2018.

[38] I. Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[39] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[40] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262, 2013.

[41] L.-Z. Guo, Z.-Y. Zhang, Y. Jiang, Y.-F. Li, and Z.-H. Zhou. Safe deep semi-supervised learning for unseen-class unlabeled data. ICML, 2020.

[42] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

[43] D. He, M. Daum, W. Cai, and M. Balazinska. Deepeverest: Accelerating declarative top-k queries for deep neural network interpretation. *Proc. VLDB Endow.*, 15(1):98–111, 2021.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[46] D. Hendrycks, M. Mazeika, and T. Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[47] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[48] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[49] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. J. Spanos. Towards efficient data valuation based on the shapley value. In K. Chaudhuri and M. Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 1167–1176. PMLR, 2019.

[50] Y. Jin, T. Zhou, L. Zhao, Y. Zhu, C. Guo, M. Canini, and A. Krishnamurthy. Autolrs: Automatic learning-rate schedule by bayesian optimization on the fly. *arXiv preprint arXiv:2105.10762*, 2021.

[51] M. Kahng, D. Fang, and D. H. Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–6, 2016.

[52] D. S. Kermany, M. Goldbaum, W. Cai, C. C. Valentim, H. Liang, S. L. Baxter, A. McKeown, G. Yang, X. Wu, F. Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[53] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[54] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[55] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[56] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[57] C.-H. Lai, D. Zou, and G. Lerman. Robust subspace recovery layer for unsupervised anomaly detection. *arXiv preprint arXiv:1904.00152*, 2019.

[58] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[59] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[60] J. Lee and G. AlRegib. Gradients as a measure of uncertainty in neural networks. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2416–2420. IEEE, 2020.

[61] D. S. Lemons and P. Langevin. *An introduction to stochastic processes in physics*. JHU Press, 2002.

[62] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[63] A. Liu, L. Reyzin, and B. D. Ziebart. Shift-pessimistic active learning using robust bias-aware prediction. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[64] M. Liu, L. Chen, X. Du, L. Jin, and M. Shang. Activated gradients for deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[65] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[66] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[67] R. B. MARIMONT and M. B. SHAPIRO. Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70, 08 1979.

[68] D. Matthew Zeiler and F. Rob. Visualizing and understanding convolutional neural networks. ECCV, 2014.

[69] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[70] M. A. Munson. A study on the importance of and time spent on different modeling steps. *ACM SIGKDD Explorations Newsletter*, 13(2):65–71, 2012.

[71] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[72] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[73] P. Perera, R. Nallapati, and B. Xiang. Ocgan: One-class novelty detection using gans with constrained latent representations. In *CVPR*, pages 2898–2906, 2019.

[74] G. Pleiss, T. Zhang, E. R. Elenberg, and K. Q. Weinberger. Identifying mislabeled data using the area under the margin ranking. *arXiv preprint arXiv:2001.10528*, 2020.

[75] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[76] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access, 2017.

[77] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*, 2018.

[78] G. Rochette, A. Manoel, and E. W. Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

[79] J. Rony, L. G. Hafemann, L. S. Oliveira, I. B. Ayed, R. Sabourin, and E. Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4322–4330, 2019.

[80] A. Ross and F. Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[81] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. Deep one-class classification. In *ICML*, pages 4393–4402, 2018.

[82] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft. Deep semi-supervised anomaly detection. In *International Conference on Learning Representations*, 2020.

[83] M. Sabokrou, M. Khalooei, M. Fathy, and E. Adeli. Adversarially learned one-class classifier for novelty detection. In *CVPR*, pages 3379–3388, 2018.

[84] M. Sakurada and T. Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pages 4–11, 2014.

[85] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[86] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[87] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[88] O. Sener and S. Savarese. Active learning for convolutional neural networks: a core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[89] Y. Shen and S. Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*, pages 5739–5748. PMLR, 2019.

[90] N. Shi, X. Yuan, and W. Nick. Semi-supervised random forest for intrusion detection network. In *MAICS*, pages 181–185, 2017.

[91] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[92] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[93] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[94] S. Sinha, S. Ebrahimi, and T. Darrell. Variational adversarial active learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5972–5981, 2019.

[95] K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.

[96] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[97] H. Song, Z. Jiang, A. Men, and B. Yang. A hybrid semi-supervised anomaly detection model for high-dimensional data. *Computational intelligence and neuroscience*, 2017, 2017.

[98] S. Swayamdipta, R. Schwartz, N. Lourie, Y. Wang, H. Hajishirzi, N. A. Smith, and Y. Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. *arXiv preprint arXiv:2009.10795*, 2020.

[99] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[100] D. M. Tax and R. P. Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[101] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, Mar. 2002.

[102] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore. Relief-based feature selection: Introduction and review. *Journal of biomedical informatics*, 85:189–203, 2018.

[103] P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 12, page 223. NIH Public Access, 2018.

[104] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1285–1300, 2018.

[105] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–3, 2016.

[106] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[107] T. Vogels, S. P. Karimireddy, and M. Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[108] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. Atomo: Communication-efficient learning via atomic sparsification. *Advances in Neural Information Processing Systems*, 31, 2018.

[109] K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2016.

[110] P. Warden. Speech commands: A public dataset for single-word speech recognition. 2017.

[111] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in neural information processing systems*, 30, 2017.

[112] W. Wu, L. Flokas, E. Wu, and J. Wang. Complaint-driven training data debugging for query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1317–1334, 2020.

[113] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun. Learning discriminative reconstructions for unsupervised outlier removal. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1511–1519, 2015.

[114] Y. Xu, L. Zhu, L. Jiang, and Y. Yang. Faster meta update strategy for noise-robust deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 144–153, 2021.

[115] H. Yong, J. Huang, X. Hua, and L. Zhang. Gradient centralization: A new optimization technique for deep neural networks. In *European Conference on Computer Vision*, pages 635–652. Springer, 2020.

[116] D. Yoo and I. S. Kweon. Learning loss for active learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 93–102, 2019.

[117] J. Yoon, S. Arik, and T. Pfister. Data valuation using reinforcement learning. In *International Conference on Machine Learning*, pages 10842–10851. PMLR, 2020.

[118] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[119] Y. Yu and C. Szepesvári. Analysis of kernel mean matching under covariate shift. *arXiv preprint arXiv:1206.4650*, 2012.

[120] Z. Yuan, J. Zhang, Y. Jia, C. Tan, T. Xue, and S. Shan. Meta gradient adversarial attack. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7748–7757, 2021.

[121] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.

[122] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang. Deep structured energy based models for anomaly detection. *arXiv preprint arXiv:1605.07717*, 2016.

[123] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[124] H. Zhang, L. Cao, Y. Yan, S. Madden, and E. A. Rundensteiner. Continuously adaptive similarity search. In *SIGMOD*, pages 2601–2616, 2020.

[125] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[126] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.

[127] Z. Zhang and T. Pfister. Learning fast sample re-weighting without reward data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 725–734, 2021.

[128] C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *SIGKDD*, pages 665–674, 2017.

[129] B. Zhu, W. Yang, H. Wang, and Y. Yuan. A hybrid deep learning model for consumer credit scoring. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 205–208. IEEE, 2018.

[130] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.