2021

Towards The Efficient Use Of Fine-Grained Provenance In Datascience Applications

Yinjun Wu
University of Pennsylvania

Towards The Efficient Use Of Fine-Grained Provenance In Datascience Applications

Abstract

Recent years have witnessed increased demand for users to be able to interpret the results of data science pipelines, locate erroneous data items in the input, evaluate the importance of individual input data items, and acknowledge the contributions of data curators. Such applications often involve the use of the provenance at a fine-grained level, and require very fast response time. To address this issue, my goal is to expedite the use of fine-grained provenance in applications within both the database and machine learning domains, which are ubiquitous in contemporary data science pipelines. In applications from the database domain, I focus on the problem of data citation and provide two different types of solutions, Rewriting-based solutions and Provenance-based solutions, to generate fine-grained citations to database query results by implicitly or explicitly leveraging provenance information. In applications from the ML domain, the first considers the problem of incrementally updating ML models after the deletions of a small subset of training samples. This is critical for understanding the importance of individual training samples to ML models, especially in online pipelines. For this problem, I provide two solutions, PrIU and DeltaGrad, to incrementally update ML models constructed by SGD/GD methods, which utilize provenance information collected during the training phase on the full dataset before the deletion requests. The second application from the ML domain that I focus on is to explore how to clean label uncertainties located in the ML training dataset in a more efficient and cheaper manner. To address this problem, I proposed a solution, CHEF, to reduce the cost and the overhead at each phase of the label cleaning pipeline and maintain the overall model performance simultaneously. I also propose initial ideas for how to remove some assumptions used in these solutions to extend them to more general scenarios.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Computer and Information Science

First Advisor

Susan B. Davidson

Keywords

Data science, Provenance

Subject Categories

Computer Sciences

TOWARDS THE EFFICIENT USE OF FINE-GRAINED PROVENANCE IN DATA SCIENCE APPLICATIONS

Yinjun Wu

A DISSERTATION
in
Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy
2021

Supervisor of Dissertation

Susan B. Davidson, Weiss Professor of Computer and Information Science, University of Pennsylvania

Graduate Group Chairperson

Mayur Naik, Professor and Graduate Chair of Computer and Information Science, University of Pennsylvania

Dissertation Committee

Zachary Ives, Adani President's Distinguished Professor and Department Chair of Computer and Information Science, University of Pennsylvania

Val Tannen, Professor of Computer and Information Science, University of Pennsylvania

James Weimer, Research Assistant Professor of Computer and Information Science, University of Pennsylvania

Todd J. Green, Principal Engineer, Amazon Web Service (AWS)

TOWARDS THE EFFICIENT USE OF FINE-GRAINED PROVENANCE IN DATA SCIENCE APPLICATIONS

Yinjun Wu

Acknowledgements

First of all, I want to thank my PhD advisor, Dr. Susan B. Davidson who treated me like her own son. During my PhD studies, she was very supportive and deeply involved in all phases of my projects, from initial discussions of potential ideas to the final stages of paper writing and conference presentation. No matter how busy she was, she was always available to meet with me and provide feedback. Without her consistently patient instruction, I would not have been able to complete this work and gain essential skills on how to conduct research projects, how to write a research manuscript and how to present in front of an audience. It was an absolutely unforgettable experience to work with her and learn from her.

In addition, I want to thank my thesis committee members, Dr. Zachary Ives, Dr. Val Tannen, Dr. James Weimer and Dr. Todd J. Green, for their valuable time and efforts in reviewing my thesis. They provided excellent suggestions on how to improve my thesis and how to present complicated research ideas in a more intuitive and clear way.

I am also grateful to Dr. Boon Thau Loo, who gave me an opportunity to interview when I was applying to graduate school in the U.S., and provided funding support towards the end of my PhD studies. Without his help, I would not have been able to work in the Penn database research group.

I also received help from many collaborators. In particular, I want to thank Drs. Abdussalam Alawini, Peter Buneman, Daniel Deutch, Edgar Dobriban, Tova Milo, Gianmaria Silvello, Val Tannen and James Weimer (their names appear alphabetically) for their valuable support in different stages of my PhD studies.

Last but not least, I want to thank my parents, Yuxiang Wu and Huaping Luo for their support and love during my PhD studies, especially during the horrible pandemic. I also want to thank my girlfriend, Chenying Zhao for staying with me and supporting me during this special period.

Abstract

TOWARDS THE EFFICIENT USE OF FINE-GRAINED PROVENANCE IN DATA SCIENCE APPLICATIONS

Yinjun Wu

Susan B. Davidson

Recent years have witnessed increased demand for users to be able to interpret the results of data science pipelines, locate erroneous data items in the input, evaluate the importance of individual input data items, and acknowledge the contributions of data curators. Such applications often involve the use of the *provenance* at a fine-grained level, and require very fast response time. To address this issue, my goal is to expedite the use of fine-grained provenance in applications within both the database and machine learning domains, which are ubiquitous in contemporary data science pipelines. In applications from the database domain, I focus on the problem of data citation and provide two different types of solutions, Rewriting-based solutions and Provenance-based solutions, to generate fine-grained citations to database query results by implicitly or explicitly leveraging provenance information. In applications from the ML domain, the first considers the problem of incrementally updating ML models after the deletions of a small subset of training samples. This is critical for understanding the importance of individual training samples to ML models, especially in online pipelines. For this problem, I provide two solutions, PrIU and DeltaGrad, to incrementally update ML models constructed by SGD/GD methods, which utilize provenance information collected during the training phase on the full dataset before the deletion requests. The second application from the ML domain that I focus on is to explore how to clean label uncertainties located in the ML training dataset in a more efficient and cheaper manner. To address this problem, I proposed a solution, CHEF, to reduce the cost and the overhead at each phase of the label cleaning pipeline and maintain the overall model performance simultaneously. I also

propose initial ideas for how to remove some assumptions used in these solutions to extend them to more general scenarios.

Table of Contents

List of Tables

List of Figures

CHAPTER 1: Introduction

Data science applications have emerged in the past few years for which fine-grained data analysis is essential. These applications include interpreting results [1], debugging suspicious items in the input [2, 3, 4], gauging the importance of data items (samples) in the data science application [5] and acknowledging the contributions of data curators and contributors [6] through data citation. To obtain results in these applications, it is essential to *efficiently* leverage *fine-grained provenance*. In the rest of this chapter, I start by briefly reviewing the history of fine-grained data provenance, and then introduce my work on automatically producing fine-grained data citations, incrementally updating machine learning models, and reducing the cost and overhead in the pipeline of cleaning label uncertainties. All of these problems involve the use of fine-grained provenance.

1.1 Review of data provenance

The question of how to capture, store and retrieve the provenance of database queries has been well studied by the database community over the last few decades. For example, [7] introduced a theoretic framework, called *how-provenance* for SPJ queries (selection-projection-join queries), extended by the follow-up work [8] for aggregate queries. Under this framework, each base relation is extended by one column, which is filled with unique identifiers (called *provenance tokens*) for each individual tuple. The provenance tokens are then propagated through database queries to the query results such that each query tuple carries a provenance expression, which is a polynomial of the provenance tokens. Other approaches to representing the provenance include *where-provenance* and *why-provenance* [9]. All of these classical provenance representations facilitate fine-grained reasoning over the provenance, and have motivated a great deal of follow-up work for maintaining and

utilizing provenance in time-critical database applications. For example, [10] uses provenance to effectively compute updated query results triggered by minor updates to the input instances using graphical representations of provenance. However, for emerging data science applications, how to efficiently employ fine-grained provenance still remains a challenge.

1.2 Using data provenance in data citation

One application of fine-grained provenance in the database domain is the problem of data citation. In this problem, the goal is to distribute credit to the contributors of data in the database by constructing citations for the query results, in which the names of contributors are included. The construction of the citation to query results relies on two factors: 1) the existing data in the database which is associated with pre-defined citation information about contributors with views of the database; and 2) how the query result can be "rewritten" using the data associated with pre-defined citation information. The first is formalized as the *citation view model* [11] while the second is the intuition behind my solutions to automatically produce data citations [12, 13], which are illustrated below.

Citation view model In the citation view model [11], each citation view consists of a *database view* and a *citation query*, which represent the data with pre-defined citation information and the pre-defined citation information itself respectively. To justify the use of this citation view model for dealing with the data citation problem, I utilize a realistic scientific database, GtoPdb [14], as an example. GtoPdb is a searchable database with information on drug targets and the prescription medicines or experimental drugs that act on them. Specifically, the following simplified schema of this database is considered (keys are underlined):

Family(<u>FID</u>, FName, Type)

FamilyIntro(<u>FID</u>, Text)

Person(<u>PID</u>, PName, Affiliation)

FC(<u>FID</u>, <u>PID</u>), FID references Family, PID references Person

FIC (FID, PID), FID references FamilyIntro, PID references Person

MetaData(Type, Value)

Each tuple in the "Family" relation represents a family of drug targets and each tuple in the "FamilyIntro" relation includes possible text as the introduction for each "Family" tuple. The other relations play a role in constructing the citation information for each "Family" tuple or each "FamilyIntro" tuple. For instance, the contributors of each family tuple could be obtained by joining the "Person" relation and the "FC" relation.

There are two reasons for the use of this citation view model. The first one is from the users' perspective: According to [15], the users of GtoPdb are typically domain experts who are very familiar with certain parts of the database, such as the "Family" relation and the "FamilyIntro" relation. Therefore, their search within this database typically starts with querying some terminology mentioned in either the "Family" relation or the "FamilyIntro" relation, followed by manual selections of appropriate records from the query result (Similar procedure also occurs in [16, 17]). Afterwards, if the users expect to give credits to the contributors of the selected data, they could click on a button provided by the DBAs to automatically obtain the citations for those data rather than query those citations from the database by themselves[1]. This would reduce the burden for those users since they may not have expertise in the entire database. The other reason for utilizing the citation view model is from the perspective of DBAs: For a large user query, such as one requesting the entire "Family" relation, DBAs may not be willing to produce very large citations which may include thousands of contributors. Instead, they would provide an abstract citation for such query, which might specify the owners of this database as the contributors for this large query result. Therefore, to facilitate such granularity control, it would be reasonable for DBAs to define two different citation views to reflect two different granularity levels. One of the citation views could target finer-grained citations, consisting of a view query which returns small amount of family tuples and a citation query retrieving the contributors of those families. In contrast, the other citation view would be designed for coarse-grained

[1] see e.g. `https://www.eagle-i.net/`

citations, comprised of a view query requesting all family tuples and a citation query to bring back the owners of the database as contributors.

My solutions I consider two different approaches to produce appropriate data citations for user query results: the Rewriting-Based Approach [12] and the Provenance-Based Approach [13]. The former uses provenance *implicitly* by adjusting traditional query rewriting using views techniques to analyze which view tuples from the pre-defined *conjunctive views* are eligible to provide citations to the query tuples in the *conjunctive query* result. This, however, fails to provide reasonable results in the case of *aggregate queries* and *aggregate views* [13], thus motivating us to come up with the second approach, i.e., the Provenance-Based Approach, which formalizes the data citation problem with the *explicit use of how-provenance*. Both approaches effectively determine which view tuples provide authorship information for each query tuple and construct formatted citations. For the Provenance-Based Approach, I also design a series of optimization strategies to minimize the overhead of querying provenance and generating citations, which achieve up to one order of magnitude speed-up relative to naive implementations. It is also worth noting that some of the techniques that I developed could also be utilized to deal with other problems, such as fine-grained access control where users are only allowed to access some pre-specified tuples [18], and the linked brushing problem in data visualization [19].

1.3 Using data provenance for incrementally updating machine learning models

The second problem which I address using fine-grained provenance is to *incrementally update machine learning (ML) models after a small number of training samples are deleted*, which is critical to efficiently evaluate the importance of one or more training samples on the (machine learning) model. Training sample importance can be quantified with many well-defined measures, such as the *Data Shapley value* [5]. A key step in evaluating this type of measure is to *remove* a subset of training samples and calculate the updated ML model

parameters. The most straightforward way to do this is to reconstruct the ML model from scratch after the samples have been deleted. However, recalculating from scratch is prohibitively expensive when the training data is frequently updated, and so the question is whether the model can be updated in real time. Unfortunately, since general machine learning models (such as neural network models) are complex, they are typically viewed as a "black box" [20] and updating them in real time is challenging. This issue also arises in other applications, e.g., refreshing the model parameters after sensitive training samples are removed (the so-called *GPDR issue* [21]), reducing the bias in statistics (e.g. *jackknife* [22]) and identifying contributors who produce the most important training samples (i.e. the ML-version data citation problem).

Therefore, the problem of incrementally updating model parameters has attracted increasing attention in the past few years. An ideal solution should effectively update *general* ML models and provide *exact updates* on the model parameters, meaning that the incrementally updated model parameters are the same as the ones reconstructed from scratch. At the same time, one would expect that such solutions would be resistant to attacks from adversaries who target recovering the data removed from the updated models (either from the model itself or from the incremental update algorithm that produces this model), which is referred to as a *model inversion attack* [23]. Unfortunately, such solutions do not currently exist (see Section 2.3 for a more detailed discussion). As the first step towards an ideal solution, I provide two approaches, i.e. PrIU (with its optimized version PrIU-opt) and DeltaGrad. The first method is designed for incrementally updating linear regression and logistic regression models, while the second method is capable of incrementally updating a more general class of models, i.e., strongly convex models. These two approaches are briefly described below.

PrIU and PrIU-opt For the problem of incrementally updating machine learning models after the deletion of a subset training samples, there is a chance to leverage the similarity between this problem and the classical *materialized view maintenance problem* in databases. One well-known solution for incrementally maintaining a materialized view after

the removal of tuples from the underlying base relations is to use the provenance semiring model [10, 24], which has been extended to handle linear algebra operators [25], i.e. the basic operators for general ML training algorithms. Hence, one natural idea is to develop solutions for propagating the deletion of training samples to ML models using provenance, specifically, using the provenance model from [25] over the linear algebraic operators in the update rule of the SGD/GD method.

However, there are some obstacles preventing the use of this provenance model for general ML models. First, this provenance model only handles linear algebra operators, ignoring the ubiquitous non-linear operators in ML training algorithms. Second, note that ML models are typically calculated iteratively with the SGD/GD method, which indicates that capturing provenance information along with such iterative computation path naively can incur non-negligible overhead, especially in large-scale applications. To deal with the first issue, I provide one solution, PrIU, for incrementally updating linear regression and logistic regression model, in which the non-linear operators appearing in the update rule of SGD/GD are linearized through *piece-wise linear interpolation* [26] so that the provenance model from [25] can be applied. The small approximation error brought by the linearization step is verified both theoretically and experimentally. To alleviate the second issue, i.e. the overhead of maintaining provenance, I provide an improved version of PrIU called PrIU-opt [27], using a series of optimization strategies. The effect of the optimization strategies is also empirically demonstrated.

DeltaGrad Since the solutions in [27] are specifically designed for ML models with simple SGD update rules, i.e. linear regression, logistic regression and possibly other *generative additive models* [28], they are hard to generalize to more complicated models, e.g. deep neural networks, which are typically trained with the gradient descent (GD) or the stochastic gradient descent (SGD) method. At each GD or SGD iteration, the major computational overhead comes from calculating the gradients on the full training samples or a mini-batch of them. Therefore, to effectively update the model parameters after the deletion of a small subset of training samples, one possible way is to reduce the overhead of evaluat-

ing the gradients at each GD or SGD iteration evaluated on the remaining training samples. I observed that such gradients could be effectively estimated with a classical optimization technique, the L-BFGS algorithm [29, 30, 31, 32]. This is significantly more efficient than computing the gradients from scratch on the remaining training samples. This idea can be applied to general ML models satisfying strong convexity, and I call this solution DeltaGrad. In this set of work, rigorous proofs are provided to show that the approximation rate achieved using the L-BFGS algorithm is close to one. Both the approximation rate and the efficiency of DeltaGrad are verified through extensive experiments over standard ML benchmark datasets, which also exhibit performance advantages over PrIU and PrIU-opt for datasets with large feature spaces.

Discussion Note that to incrementally update a strongly convex model after a deletion, an alternative way would be to start from the model constructed on the full training set and continue running GD or SGD on the remaining training samples for t iterations (t is a small number) until convergence, which is considered in [33] and is referred to as the *online method* hereafter. The resulting model incrementally updated in this manner is guaranteed to be exactly the same as the one retrained from scratch on the remaining training samples. This is because the local minimum (also the global minimum) is unique for strongly convex models, and the model differences are not significant before and after the deletions of a small number of training samples.

However, the online method has two limitations. First, the online method may suffer from a *model inversion attack* [23], in which the adversary attempts to extract the *deleted* training samples from the updated models. This can occur especially when the adversary has *white-box* access to the models, meaning that the adversary has full knowledge of the models, such as the model parameters, model type, learning algorithms, hyper-parameters and even the remaining training samples (see Section 2.3 for more details). Specifically, if only one training sample is removed, then under the *white-box model inversion attack*, the adversary is able to reconstruct the removed training sample by utilizing the gradients evaluated on this sample, which can be recovered through reversing the online method (see

Section 6.1.2.2 for more details). In contrast, both PrIU and DeltaGrad are resistant to the *white-box model inversion attack*. Intuitively speaking, these two approaches mimic how the machine learning models are retrained from scratch on the remaining training samples, but accomplish it in an efficient way. This can thus "erase" the footprint of the deleted training sample from *all* GD or SGD iterations so that the adversary is unable to retrieve any information about the deleted training sample from those GD or SGD iterations, thus safeguarding the removed training sample.

The second limitation of the online method is its failure to accurately update more complicated models. In other words, those complicated models incrementally updated by the online method will be significantly different from the ones retrained from scratch on the remaining training samples. Detailed discussions about this limitation are provided in Section 6.1.2.3.

1.4 Using data provenance to reduce the cost of cleaning label uncertainties

The third part of my thesis work focuses on reducing the time overhead and the cost in the pipeline of cleaning uncertain labels for improving ML model quality, which relies on the my solution for incrementally updating ML models, e.g. DeltaGrad. This application arises due to the need for high-quality labels, which is critical to produce high-performance ML models. However, collecting such labels for all training samples is very time consuming and expensive. This is because in some domains, such as medical imaging, automatic labeling tools are error-prone [34] and only human annotators with domain knowledge (e.g. doctors or physicians) can provide reliable labels. To deal with this label scarcity problem, one can leverage the labeling functions in Snorkel [35] to automatically derive *probabilistic labels* (or *weak labels*) for large amount of training samples. However, those probabilistic labels may not be perfect and even inaccurate [36, 37, 38], thus hurting the model quality [39]. Therefore, it is essential to perform additional cleaning operations on those labels.

The label cleaning pipeline is typically *iterative* [40, 2], and requires multiple rounds (see Figure 1.1, loop labeled (1)). First, given a *cleaning budget* B, the top-B influential training samples with probabilistic labels are selected (the *sample selector phase*). Second, for those selected samples, cleaned labels are provided by human annotators (the *annotation phase*). Third, the updated ML model is calculated using the updated training set (the *model constructor phase*), and returned to the user. If the resulting model performance is not good enough, the process is repeated with an additional budget B'. Otherwise, it is deployed. Note that since each of these phases may be performed *repeatedly*, it is important that they be as efficient as possible. It is also noteworthy that for some applications—such as the medical image classification task—it is essential to have multiple human annotators for label cleaning to alleviate their labeling errors [41] in the *annotation phase*, thus incurring substantial time overhead and financial cost. Therefore, I propose a solution called CHEF (CHEap and Fast label cleaning) [42] (the extended technical report could be found in [43]), to reduce the time overhead and cost of the label cleaning pipeline and simultaneously enhance the overall model performance.

Specifically, in the *sample selector phase*, given a fixed cleaning budget B, I propose a method, Infl, a variant of influence function [44], to prioritize the most influential training samples for cleaning. In comparison to the classical methods for selecting unlabeled training samples for labeling, such as the active learning methods [45], and the ones for cleaning noisy labels, such as O2U [46], Infl can not only suggest which training samples to be cleaned, but also *automatically* derive the potentially clean labels, which are close to or even better than the human annotated labels. Therefore, utilizing those labels as one alternative labeler in the *annotation phase* can significantly reduce the human annotation efforts.

In addition, I notice that employing Infl comes at a price due to its overhead in evaluating the gradients for each individual training sample. Therefore, by assuming strong convexity on the model type, I develop Increm-INFL in the *sample selector phase* to filter out most of the uninfluential training samples early by estimating the perturbation bound on their influence but without explicitly evaluating their influence. Furthermore, to accelerate the

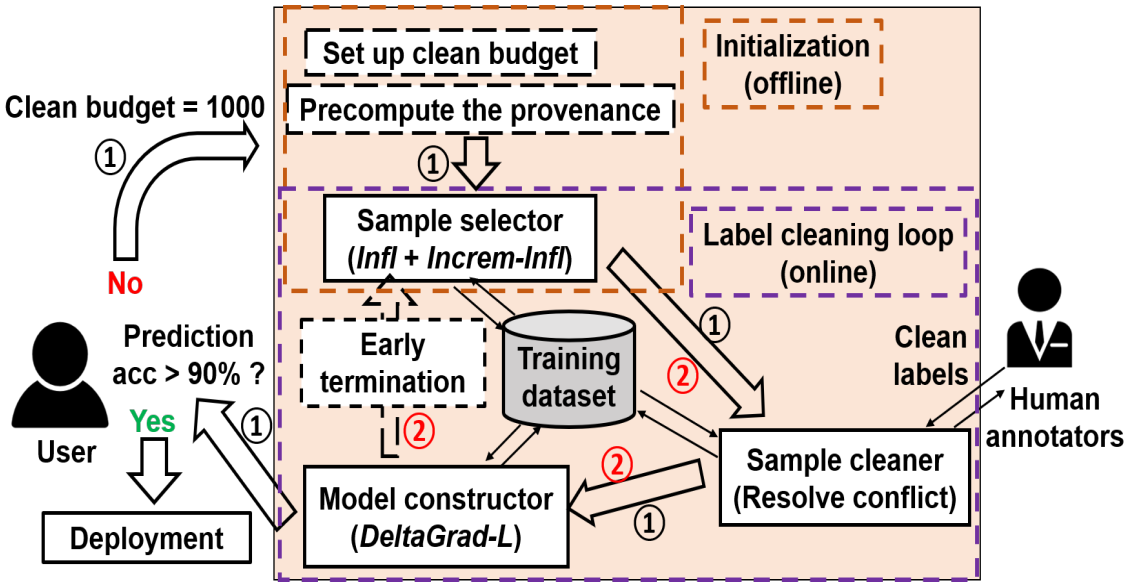


Figure 1.1 – The iterative pipeline of cleaning uncertainties from the labels of training set.

model constructor phase, DeltaGrad is adapted (referred to as DeltaGrad-L) to incrementally update models after the probabilistic labels of the most influential training samples are cleaned, which requires that the updated models are strongly convex. Last but not least, due to the reduced running time in the *sample selector phase* and the *model constructor phase*, I can allow the human annotators to interact with the entire system for multiple times and every time they can choose to clean the labels of Top-b influential training samples, in which b could be smaller than cleaning budget. This can possibly lead to better overall model performance and early termination once the users' expected model performance is reached before the cleaning budget is exhausted.

1.5 Future extensions for DeltaGrad and CHEF

The aforementioned solutions, DeltaGrad and CHEF, can be extended in various ways, which is left as future work. First, as introduced in Sections 1.3 and 1.4, DeltaGrad and CHEF rely on an assumption about the model class, which may not hold in practice. Therefore, an immediate extension to those two solutions is to generalize them to handle more complicated machine learning models, such as neural network models. Specifically, for DeltaGrad, I

have initial ideas for simultaneously generalizing it to neural network models and guarding against model inversion attacks to some degree by leveraging the online method and using some additional properties of over-parameterized neural network models (see Section 6.1 for details).

To extend CHEF to deal with general neural network models, it is necessary to extend Increm-INFL. Recall that Increm-INFL is capable of estimating the perturbation bound on all the training sample influence such that uninfluential training samples are removed early during the process of determining the most influential training samples. To facilitate the estimation of the above perturbation bound when general machine learning models are used, I observe that this perturbation bound depends on a Hessian-vector product, which could be evaluated with an extended L-BFGS algorithm [47] for general machine learning models. (see Section 6.2.1).

In addition to the above, CHEF can be extended in two more ways. First, since the probabilistic labels on the uncleaned training samples may depend on the existing labeled training samples according to some existing weak-supervised learning solutions (see e.g., [48]), then as the labels of more and more training samples are cleaned, it would be reasonable to update the probabilistic labels for the remaining training samples accordingly to eliminate possible labeling bias caused by the labeled training samples at the initial stage [49]. I therefore propose to integrate CHEF with weakly supervised learning in a tighter way, in which the probabilistic labels are also dynamically updated to reflect the increasing number of cleaned training samples for better overall model performance. To accomplish this, it may be necessary to develop efficient solutions to estimate the magnitude of the updates on the probabilistic labels (see Section 6.2.2).

Furthermore, to minimize the manual labeling cost for certain learning tasks, one could also turn to semi-supervised learning, which uses both labeled and unlabeled training samples in the model training process [50, 51]. To achieve better overall model performance, there are some recent efforts in combining active learning and semi-supervised learning such that the labeled training samples used for semi-supervised learning are appropriately selected,

rather than randomly selected [52, 53, 54, 55]. As I will illustrate in Chapter 5, Infl in CHEF can be regarded as an alternative to active learning methods for identifying which training samples to be cleaned even when the uncleaned training samples are all unlabeled (even without probabilistic labels). Therefore, by following the framework in [56], I also propose to combine CHEF with semi-supervised learning with slight modifications on Infl (see Section 6.2.3).

1.6 Summary and Roadmap

In summary, my dissertation addresses the problem of efficiently using *fine-grained provenance* in data science applications. It consists of four interrelated parts:

1. My work on *data citation*, which effectively generates *fine-grained* citations to individual output query tuples by *implicitly* or *explicitly* using data provenance.

2. My work on efficiently updating ML models constructed by GD/SGD methods after the deletion of a small subset of training samples, PrIU and DeltaGrad, which leverages provenance information collected during the training phase prior the deletion of any training samples. The technique is also useful for efficiently evaluating the importance of subsets of training samples at a *fine-grained level*.

3. My solution to reduce the time overhead and cost of cleaning label uncertainties, which includes 1) prioritizing the most influential training samples for cleaning and suggesting potentially clean labels with Infl, 2) filtering out uninfluential training samples early with DeltaGrad-L, 3) incrementally updating the models with Increm-INFL after the most influential training samples are cleaned, 4) and redesigning the entire pipeline to facilitate better overall model quality and early termination.

4. A discussion of possible future extensions for DeltaGrad and CHEF. One immediate extension is to generalize these two solutions to handle more general machine learning models. Another is to integrate CHEF with weakly- and semi-supervised learning.

The rest of the dissertation is organized as follows. In Chapter 2, I summarize related work, which is followed by a discussion of my solutions for the problem of data citation in Chapter 3. I then discuss how to incrementally update machine learning models with PrIU and DeltaGrad in Chapter 4, and how to reduce the time overhead and cost in the pipeline of cleaning label uncertainties with CHEF in Chapter 5. Possible future extensions for DeltaGrad and CHEF are presented in Chapter 6. Finally, I conclude and discuss future work in Chapter 7.

CHAPTER 2: Related work

In this chapter, I summarize related work in several different areas, starting with classical data provenance frameworks, followed by a discussion of related work in each of the other provenance-related problems that I have worked on: Reasoning about fine-grained data citations, and evaluating the importance of training samples to ML models using provenance. I close by discussing prior work related to the proposed work.

2.1 Data provenance

In the database domain, there are several classical theoretical frameworks [7, 9, 8] that can facilitate fine-grained provenance analysis. For example, [7] introduced a framework called *how-provenance* for select-project-join (SPJ) queries, which was extended in follow-up work [8] to aggregate queries. Under this framework, each base relation is extended by one column, filled with unique identifiers (called provenance tokens) for each individual tuple. The provenance tokens are then propagated through the database queries to query results such that each query tuple carries a provenance expression, composed of a polynomial of the provenance tokens, in which each monomial (i.e. the provenance token sub-expression concatenated by "*") represents the *joint* tuples after the join operations and "+" concatenates those monomials from the duplicated joint tuples after the projection operation. Then whether a certain base relation tuple t_s contributes to the construction of a certain query tuple t_q can be determined by whether or not the unique provenance token of t_s appears in the provenance expression of t_q. A recent paper [25] also extends the provenance semiring model to handle linear algebra operators, which are the basic operators for general ML algorithms.

2.2 Generating fine-grained data citations with fine-grained provenance

I now discuss work related to reasoning about fine-grained data citations. First, I provide a high-level overview of the data citation problem. Then I discuss classical query-rewriting-using-views methods, which can be used to model the data citation problem.

Data citation. Principles for data citation have been proposed within the digital library community by CODATA [57] and FORCE 11 [58], which include: 1) identification and access to the cited data; 2) persistence of the cited data; and 3) completeness of the reference [59, 60, 61, 62]. The community also recognized the importance of citations to aggregate data [57], as have various scientific communities [63, 64, 65]. Early solutions to data citation were proposed by digital libraries experts, which, however, failed to handle the versioning problem of data citations and could not automatically generate citations for arbitrary snippets of information, which are useful for human understanding, thus violating *persistence* and *completeness*. The data citation problem has recently captured the attention of database researchers, who formulated this problem as a computational challenge [66, 67] and defined a model of *citation views* in [11]. In additions to my solutions to automatically produce fine-grained data citations, i.e. the Rewriting-Based Approach [12] and the Provenance-Based Approach [13], the automatically generated data citations could be provided to users via a user interface [68] and the connections between data citations and data provenance are also discussed in [69]. To facilitate the practical use of the Rewriting-Based Approach and Provenance-Based Approach, [70] discusses how to adjust the scientific databases with data citation support such that the citations to the query result against those databases can be detected by Google Scholar.

Query rewriting using views. Query rewriting using views has been applied to many data management problems, in particular query optimization and data integration [71]. Query rewriting using views is centered around the notion of *containment* and *equivalence* of queries [71]. Specifically, a query $Q1$ is *contained* in a query $Q2$, denoted $Q1 \sqsubseteq Q2$, iff for

any database instance D, $Q1(D) \subseteq Q2(D)$. $Q1$ is *equivalent* to $Q2$, denoted $Q1 \equiv Q2$, iff $Q1 \sqsubseteq Q2$ and $Q2 \sqsubseteq Q1$. The problem of Query rewriting using views has been extensively studied in the context of conjunctive queries [72, 73, 74, 75] as well as aggregate queries [76, 77]. Various algorithms have been designed to rewrite aggregate queries. For example, [78, 79] provide algorithms for determining whether a materialized view is usable for answering an aggregate query by considering both conjunctive and aggregate views. In [80], an algorithm is given to handle nested subqueries and multidimensional aggregations in queries and views. However, only standard aggregate functions (e.g. SUM, COUNT) are considered in [80, 78, 79]; general aggregate functions (such as user defined aggregate functions) cannot be used. The problem of general aggregate functions is considered in [81], and [82] bridges the gap between theory and practice by providing implementation suggestions. Classical query rewriting using views methods involve *coarse-grained* reasoning on the query schema and view schema. In contrast, in the data citation problem, it is indispensable to analyze the dependency between database views and queries at the tuple-level, *given* a database instance, thus necessitating *fine-grained* analysis.

2.3 Incrementally update machine learning models with provenance

The problem of incrementally updating machine learning models has attracted a lot of attention in both the machine learning and the database communities. I review this work, and then familiarize readers with the classical BFGS algorithm and its variants that our solution, DeltaGrad heavily relies on. Afterwards, I also briefly introduce the concept of the model inversion attack that the solutions to incrementally update models need to address. This is then followed by some recent work on in-database learning, an application where incremental updates on ML models may also occur. This section then ends with related work on online learning, which is also relevant to the problem of incrementally updating machine learning models.

Incrementally updating machine models. To deal with the incremental update problem for machine models, there is prior work on estimating the updated model parameters without retraining from scratch. It is worth noting that different solutions may impose different requirements on the incrementally models, in particular, their difference with respect to the models retrained from scratch. To our knowledge, there exist two different such requirements in literature. The first one is that with some probabilistic guarantees, the incrementally updated models cannot be distinguished from the models retrained from scratch, which is relevant to the notion of differential privacy [83]. Typical solutions depending on this requirement include Descent-to-delete [33], which, incrementally updates models by utilizing the online method (to be introduced later in Chapter 6.1.2) first and then adds noise afterwards. In contrast, the second requirement does not involve any probabilistic claims, which, instead, requires that the incrementally updated models are *deterministically* as similar to the retrained models as possible. In my dissertation, I primarily focus on developing solutions (to be described later) satisfying the second requirement. In what follows, for the state-of-the-art solutions within the ML community that also fulfill the second requirement, they are visually compared against my solutions in Figure 2.1 along two axes: The complexity of the models to be dealt with (the x-axis); and the similarity between the incrementally updated models and the expected models, i.e. the models reconstructed from scratch (the y-axis). The ideal solution for incrementally updating models is located at the top right corner in the figure, which should effectively update *general* ML models and provide *exact updates* on the model parameters, meaning that the incrementally updated model parameters are the same as the ones reconstructed from scratch.

Unfortunately, there remain obvious gaps between the state-of-the-art and the ideal solution. For example, [44] proposes an *influence function* for general ML models. However, it has several disadvantages, including the expense of computing the Hessian matrix of the objective function, poor robustness [84], and poor accuracy of estimating the updated model parameters after deleting multiple samples [27]. Therefore, this method is placed in the middle at the right hand side of the figure. In contrast, other state-of-the-art works

focus on incrementally updating some *specific*, fairly simple, ML models, and therefore lack complexity. For example, [85, 86, 87, 88, 89, 90, 91, 92] resolve the issue of incrementally updating simple linear ML models, including linear regression, naive Bayes, nearest neighbor and support vector machine models, and are therefore located near the top left corner in the figure. In [93], a modified K-means clustering algorithm using a quantization-based solution provides theoretical guarantees on the stability of the model parameters (i.e. the centroids as the result of the K-means clustering algorithm) when deletions of samples occur, therefore avoiding retraining. Since the K-means model is more advanced than a linear model, and [93] exactly updates the model parameters, this solution is located near the top middle of the figure. However, current state-of-the-art works either fail to robustly and accurately update the model (e.g. [44]), target only simple models, or require changes to the training algorithms (e.g. [93]), and are therefore not appropriate for ML models constructed by some of the most widely used methods: gradient descent (GD) and its variants, i.e. stochastic gradient descent method (SGD) and mini-batch gradient descent method (mb-SGD for short).

It is also worth noting that other than my solutions, PrIU and DeltaGrad, we can also consider using online method for incrementally updating machine learning models that are constructed by GD or SGD method. Intuitively speaking, after the deletions of certain training samples, online method continuously runs GD or SGD iterations by utilizing the remaining training samples and the iterations start from the models that are constructed on the full training dataset. As I will introduce in Section 6.1.2, arbitrary convex models can be incrementally updated by online method and the resulting models are the same as the ones reconstructed from scratch. Therefore, in Figure 2.1, online method locates on the top boundary and slightly to the right of DeltaGrad. However, as I will introduce in Section 6.1.2.2, one disadvantage of online method prevents its use in some scenarios, i.e., its failure to defend against certain type of model inversion attack [94].

The BFGS algorithm and its variant, the L-BFGS algorithm. There have been many studies in the past few decades on utilizing Quasi-Newton methods for large-scale optimization problems, which leverage the second order information of the objective func-

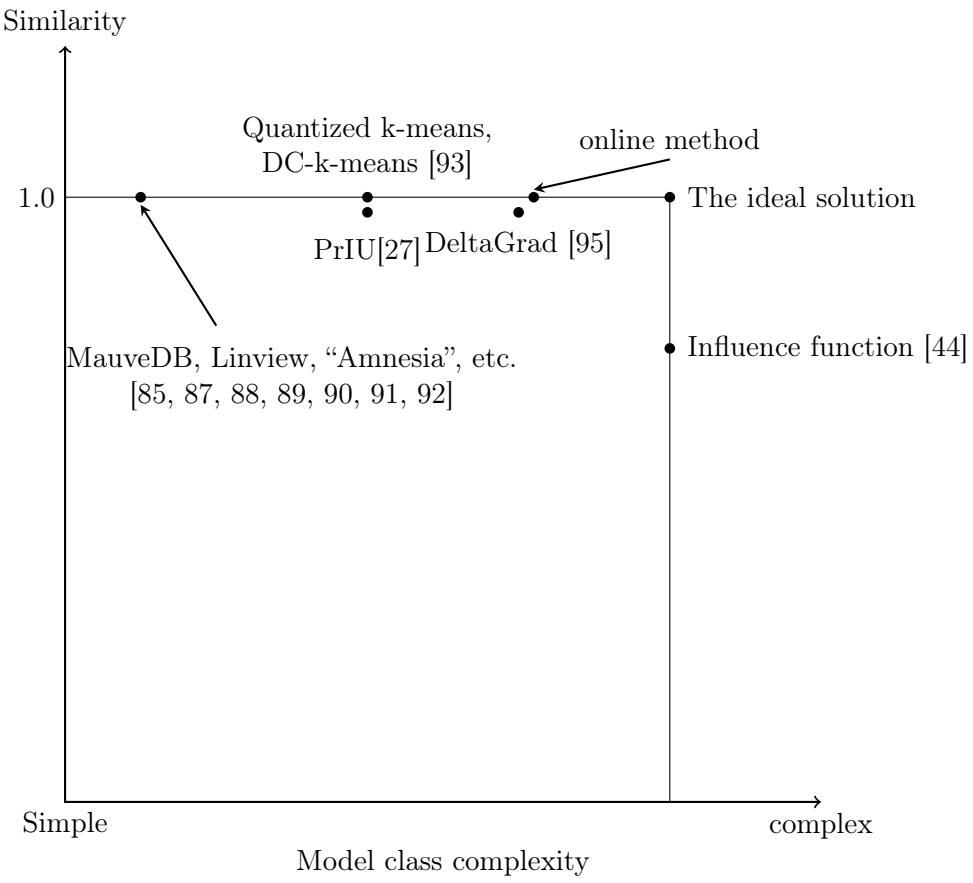


Figure 2.1 – Comparison between PrIU, DeltaGrad and the state-of-the-art works. The x-axis represents the complexity of ML model classes that each incremental update method can handle, which is determined by the ML model representation (e.g. quantified by the model parameters) and the way to obtain the model parameters (e.g. closed-form solutions VS iterative methods). The y-axis represents the similarity between the resulting updated models by each incremental update method and the one obtained by retraining from scratch.

tions. One such representative is the BroydenâĂŞFletcherâĂŞGoldfarbâĂŞShanno (BFGS) algorithm and its variant, the Limited-memory-BFGS (L-BFGS) algorithm, which effectively update the model parameters and simultaneously reduce the overhead of maintaining and computing the hessian matrix. The earliest version of the BFGS algorithm dates back to the 1970s [96, 97] which updates the model parameters with the following update rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \mathbf{B}_t^{-1} \nabla F(\mathbf{w}_t),$$

in which α_t is the step size at the t_{th} iteration, $\mathbf{w}_t$ is the model parameter to be derived by minimizing the objective function $F(\mathbf{w}_t)$ and $\mathbf{B}_t$ is the approximated hessian matrix of the objective function $F(\mathbf{w}_t)$. $\mathbf{B}_t$ is also updated at each iteration and it is usually assumed that the changes of $\mathbf{B}_t$ at each iteration are low-rank, e.g. rank-one or rank-two updates, i.e.:

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\mathbf{y}_t \mathbf{y}_t^T}{\mathbf{y}_t^T \mathbf{s}_t} - \frac{\mathbf{H}_t \mathbf{s}_t \mathbf{s}_t^T \mathbf{H}_t}{\mathbf{s}_t^T \mathbf{H}_t \mathbf{s}_t}$$

where $\mathbf{s}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$ and $\mathbf{y}_t = \nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t)$.

To reduce the overhead in limited memory scenarios, [29] proposed the limited-memory-BFGS algorithm, which avoids maintaining the approximated hessian and estimating the matrix-vector product $\mathbf{B}_t^{-1} \nabla F(\mathbf{w}_t)$ directly. Details of the L-BFGS algorithm will be presented in Algorithm 8 in Section 4.3.1. There have been a series of studies on exploring the approximation rate and convergence rate of the BFGS and L-BFGS algorithms [98, 29, 32, 99, 31, 100, 30], ideas from which are also used in our theoretical analysis.

Model inversion attack The model inversion attack in general refers to an attack from an adversary who has no access to certain training samples but attempts to reconstruct those samples from the machine learning model [23]. This attack, however, has a slightly different meaning in the context of updating models triggered by the removal of sensitive training samples, in which the adversary targets extracting the *deleted* training samples from the updated models [94]. Depending on how much information the adversary obtains, the model inversion attack could be either *black-box* or *white-box* [101]. For a black-box attack, the adversary can only obtain the model output given an input training sample without knowing any more information about the models. In contrast, for a white-box attack the adversary has full knowledge of the model, such as the model parameters, the model types, the remaining training samples after the deletion requests, the learning algorithm (including the hyper-parameters) and even the algorithm for updating the models. Therefore, the white-box set-up is easier for the adversary to launch the attack with respect to the black-box set-up. However, the black-box assumption is more reasonable in practice, since state-of-the-art online model APIs regard the deployed models as black boxes to which users can

only have oracle access[1].

In-database learning According to a recent survey released by Kaggle[2], the majority of the practical data science tasks involve relational data. The mainstream approaches to conducting machine learning tasks over such type of data is to export those data from the relational databases first, followed by importing them into the modern machine learning libraries, such as Pytorch [102] and Tensorflow [103]. However, this can lead to severe performance issues since it is essential to query the training data from the relational database, which may require to join multiple relational tables, leading to huge materialized query results. One possible way is *structure agnostic*, in which the DBMSs and machine learning libraries are loosely integrated by representing the functions in those libraries as the user-defined aggregate functions in DBMSs (e.g., see MADlib [104] and the state-of-the-art machine learning engine in Teradata SQL [105] and Microsoft SQL server[3]). The other possible way is to leverage the relational structures in DBMSs and this type of methods is referred to as *structure-aware* methods. For example, some linear algebraic computations conducted by the machine learning libraries could be transformed to SQL aggregate queries, which could be pushed past the join operations to avoid materializing large query results representing the training data [106, 107, 108]. This could significantly accelerate the training process of some simple machine learning models inside DBMSs, such as the linear regression model. For this type of *structure-aware* methods, the above aggregate query results representing linear algebraic computations could be cached for incrementally maintaining the models. However, recall that only the simple models are handled by the *structure-aware* methods. Therefore, the above techniques for incrementally maintaining models within DBMSs may not be suitable for general models, especially when the *structure agnostic* methods are used [109]. We would expect that this problem could be fixed when the approaches to incrementally updating machine learning models, such as DeltaGrad, become

[1] see e.g., Google prediction API:
`https://cloud.google.com/ai-platform/prediction/docs/reference/rest/v1/projects/predict`

[2] `https://www.kaggle.com/surveys/2017`

[3] `https://docs.microsoft.com/en-us/sql/machine-learning/sql-server-machine-learning-services?view=sql-server-ver15`

more mature in the future.

On-line learning On-line learning concerns a learning task where the training data arrive in a sequential order. Different from the traditional off-line (or batch) machine learning methods where the full training set is available during the whole training process, on-line learning aims at constructing a model to maximize the prediction accuracy given the current observation and the previous predictions on the prior observations [110], which requires to continuously update models given more and more newly added training samples. This indicates that the on-line learning methods do not take into account the deletions of training samples. However, as revealed in [33], the spirit of on-line learning, i.e., learning over a sequence of training samples, could also be applied for incrementally updating strongly convex models after the deletions of small amount of training samples. Specifically, this can be accomplished by starting from the model constructed on the full training set and continuing running SGD for a few epochs on the remaining training samples after the deletion requests, finally resulting in an exactly updated model, which utilizes the uniqueness of the local minimums of the strongly convex models. However, as discussed in Section 1.3, this method 1) may suffer from the white-box model inversion attack [23] when the adversary knows everything about the models; 2) cannot be generalized to more complicated models, such as over-parameterized models, which fully memorize all training samples, including samples with random labels.

2.4 Related work on cleaning label uncertainties for machine learning models

Finally, I review the literature on cleaning label uncertainties for machine learning models, which is comprised of the related work on some general data cleaning problems in the machine learning pipeline, followed by some extra related work on how to prioritize the most influential training samples for cleaning their labels. I then conclude this section with the literature review on semi-supervised learning and weakly supervised learning, which

paves the way to the future work on integrating CHEF with those two learning frameworks.

Data cleaning for ML models Diagnosing and cleaning errors or noises in training samples has attracted considerable attention [46, 111], and is typically addressed iteratively [40, 112, 2]. For example, the authors of [46] observed that the noisily labeled samples were memorized by the model in the overfitting phase, which can be detected through transferring the model status back to the underfitting phase. [111] identifies and fixes the noisy labels through jointly analyzing how probable one noisy label is flipped by the human annotators and how this label update influences the model performance. However, it explicitly assumes that the noisy labels are either 1 or 0, thus not applicable in the presence of probabilistic labels. The approach in [2] detects errors in both feature values and labels; But it explicitly assumes that the uncleaned samples are harmful and thus excluded in the training process, we follow the principle of [35] by "including" the training samples with uncertain labels in the training phase.

Detecting the most influential training samples with uncertainties As discussed in [112], it is important to prioritize the most influential training samples for cleaning. This can depend on various influence measures, e.g., the uncertainty-based measures in active learning methods [45], the influence function [44], the data shapley value [113], the loss produced by neural network models [46, 114], etc. However, to our knowledge, none of these techniques can be used to automatically suggest possibly cleaned labels, apart from [115]. Furthermore, the applicability of [115] is limited due to its poor scalability and some of the above methods (including [115]) are not applicable in the presence of probabilistic labels and the regularization on them.

Semi-supervised learning Similar to active learning methods, semi-supervised learning is also a methodology for dealing with the label scarcity issues, which employs both the labeled training samples and the unlabeled training samples for certain machine learning tasks without relying on extra labels provided by human annotators [50, 51]. However, as mentioned above, semi-supervised learning also depends on certain amount of labeled samples. Therefore, given a training set without any labeled samples, to initialize the

semi-supervised learning tasks, it is crucial to determine appropriate samples rather than randomly selected samples for labeling such that the performance of the resulting model could be optimized [53], thus motivating a line of research on integrating semi-supervised learning with active learning, e.g., [53, 55, 52, 54, 116, 56]. Among those studies, [56] provides a more general framework for handling arbitrary machine learning models and adjusts the metric in active learning for selecting the most informative unlabeled training samples for labeling such that it could be more coherent with the objective function of the semi-supervised learning method. Considering my proposed method, INFL, is an alternative to the active learning method, it is thus worth integrating INFL with the semi-supervised learning method for gaining better model quality, which is discussed in Section 6.2.

Weakly supervised learning Similar to semi-supervised learning, weakly supervised learning also aims at dealing with the lack of enough high-quality labels in practice by utilizing the unlabeled training samples, but from a different perspective. Specifically, weakly supervised learning automatically generates lower-quality probabilistic labels for unlabeled training samples by leveraging some heuristics (named as *labeling functions* [35]) provided by domain experts, which are then included as part of the training set for the learning tasks. Despite the low cost of obtaining those probabilistic labels, their quality may be not ideal [49, 36, 37, 38] due to the imperfect labeling functions. To address this problem, there were some recent research efforts on combining active learning and weakly supervised learning. Intuitively speaking, the mechanism of active learning is employed to let the human annotators refine some probabilistic labels (e.g. the most informative ones) and those refined labels could then be leveraged to generalized to other training samples to repair their probabilistic labels [49, 117, 118]. This process could proceed iteratively until the satisfactory model performance is reached. Note that our method, CHEF, only loosely incorporates weakly supervised learning since the refinement of a small amount of probabilistic labels (identified by INFL) is not propagate to the remaining probabilistic labels. As I will discuss in Section 6.2.2, similar to active learning, accomplishing the tight integration between CHEF and weakly supervised learning is also possible, which is left as part of the future

work.

CHAPTER 3: Reasoning about fine-grained data citations with provenance

In this chapter, I will introduce how I handle the data citation problem in two different ways, i.e. Rewriting-based approaches and Provenance-based approaches, which is started by introducing the concept of the *citation view model*. In the remainder of this chapter, the following simplified GENCODE [63] database schema is used to illustrate the key concepts:

Gene(GID, Name, Type)
Gene2Contributor(GID, Person) GID references Gene
Transcript(TID, Name, Type, GID) GID references Gene
Transcript2Contributor(TID, Person) TID references Transcript
Exon(EID, Level, TID), TID references Transcript
Exon2Contributor(EID, Person), EID references Exon

This schema is composed of six relations, including the relation "Gene", "Transcript" and "Exon", each of which is associated with one relation to record the corresponding contributors. Given this schema, a simplified instance of this database instance is presented in Table 3.1-3.5.

3.1 Preliminary

3.1.1 Citation view models

As the first step toward the full-fledged solutions to produce fine-grained citations for general user queries, I defined the *citation view model* [11], composed of view queries and citation queries. The definition of the citation view model is provided as below:

Definition 1. *A citation view is a tuple* (V, C_V) *where:*

Table 3.1 – Instance of relation *Exon*

	EID	Level	TID
t_{e1}	1	1	1
t_{e2}	2	3	2
t_{e3}	3	2	2
t_{e4}	4	2	2

Table 3.2 – Instance of relation *Exon2Contributor*

	EID	Person
t_{ec1}	1	Joe
t_{ec2}	2	David
t_{ec3}	3	Mark
t_{ec4}	4	Robert

Table 3.3 – Instance of relation *Gene*

	GID	Name	Type
t_{g1}	1	TF	TEC
t_{g2}	2	FH	rRNA
t_{g3}	3	RP1	rRNA
t_{g4}	4	IYD	rRNA
t_{g5}	5	EPN	mRNA

Table 3.4 – Instance of relation *Gene2Contributor*

	GID	Person
t_{gc1}	1	Jane
t_{gc2}	2	David
t_{gc3}	3	Chris
t_{gc4}	4	Tom
t_{gc5}	5	Joe

Table 3.5 – Instance of relation *Transcript*

	TID	Name	Type	GID
t_{t1}	1	MB-203	TEC	1
t_{t2}	2	PC-203	rRNA	2
t_{t3}	4	HP-218	rRNA	2
t_{t4}	5	TP-208	rRNA	3

Table 3.6 – Instance of relation *Transcript2Contributor*

	TID	Person
t_{t1}	1	David
t_{t2}	2	Jane
t_{t3}	4	Mark

1. *V is the view definition of the form* $\lambda \mathtt{X}.\mathtt{V}(\mathtt{Y}) :- \mathtt{Q}$;

2. *C_V is the citation query of form* $\lambda \mathtt{X}.\mathtt{C_V}(\mathtt{Y'}) :- \mathtt{Q'}$;

In this definition, V and C_V are parameterized conjunctive Datalog queries possibly with aggregations but no negations, in which Y (Y' resp.) is a set, including either variables from the predicates in Q (Q' resp.) or *aggregated terms* which are the terms utilizing aggregate functions over the variables coming from the body of Q. The parameter X in the lambda term λX is a set of variables (not aggregated variables) satisfying $X \subset Y$, which is optional in the definition of V and C_V. The role of λX is to control the granularity of the citations annotated in the instance of V, where the view tuples with the same values in the variables from X share the same citations. If the term λX does not exist, then all the view tuples in the view instance share the same citations.

Given a user query Q and the corresponding query instance, the first step is to check each individual query tuple to which view tuples are responsible for its construction, followed by extracting snippets of the citation information associated with those responsible view tuples by issuing the citation queries, which are then used to construct citations for the query tuples by some predefined formats, e.g. the JSON format, in the *citation function*.

Example 1. *I can define some citation views for the simplified GENCODE databases, in which the view definitions are presented as below:*

$$
\begin{aligned}
\lambda G.V_1(G, Ty) \quad & :- Gene(G, N, Ty), G \leq 2 \\
V_2(Ty, COUNT(G)) \quad & :- Gene(G, N, Ty), Ty = \text{`}rRNA\text{'} \\
V_3(G, COUNT(T)) \quad & :- Transcript(T, N, Ty, G), T \leq 2 \\
V_4(G, N) \quad & :- Transcript(T, N, Ty, G), T \geq 2 \\
\lambda Ty2.V_5(G, N, Ty2) \quad & :- Gene(G, N, Ty1), Transcript(T, N', Ty2, G') \\
& \quad , G = G', T \geq 4 \\
V_6(T1, E, G1, L) \quad & :- Transcript(T1, N1, Ty1, G1), Exon(E, L, T2) \\
& \quad , T1 = T2, E \leq 2 \\
V_7(G1, COUNT(T1)) \quad & :- Transcript(T1, N1, Ty1, G1), Exon(E, L, T2)
\end{aligned}
$$

$$, T1 = T2, L \leq 2$$

$$V_8(G1, MAX(L), COUNT(E)) : -Transcript(T1, N1, Ty1, G1),$$
$$Exon(E, L, T2), T1 = T2$$

The instances of the views defined above are provided in Table 3.7-3.8 and the corresponding citation queries are presented as below:

$$\lambda G.C_{V_1}(G, Array\_agg(P)) \quad : -Gene(G, N, Ty), Gene2Contributor(G', P)$$
$$, G = G', G \leq 2$$

$$C_{V_2}(Array\_agg(P)) \quad : -Gene(G, N, Ty), Gene2Contributor(G', P)$$
$$, G = G', Ty = `rRNA$$

$$C_{V_3}(Array\_agg(P)) \quad : -Transcript(T, N, Ty, G),$$
$$Transcript2Contributor(T', P), T = T', T \leq 2$$

$$C_{V_4}(Array\_agg(P)) \quad : -Transcript(T, N, Ty, G),$$
$$Transcript2Contributor(T', P), T = T', T \geq 1$$

$$\lambda Ty2.C_{V_5}(Ty2, Array\_agg(P)) \quad : -Gene(G1, N1, Ty1), Transcript(T, N, Ty2, G2),$$
$$Transcript2Contributor(T', P)$$
$$, T = T', G1 = G2, T \geq 4$$

$$C_{V_6}(Array\_agg(P)) \quad : -Transcript(T1, N1, Ty1, G1), Exon(E, L, T2),$$
$$Exon2Contributor(E', P), E = E', T1 = T2$$
$$, E \leq 2$$

$$C_{V_7}(Array\_agg(P)) \quad : -Transcript(T1, N1, Ty1, G1), Exon(E, L, T2),$$
$$Transcript2Contributor(T3, P)$$
$$, T2 = T3, T1 = T2, L \leq 2$$

$$C_{V_8}(Array\_agg(P)) : -Transcript(T1, N1, Ty1, G1), Exon(E, L, T2),$$
$$Transcript2Contributor(T3, P), T2 = T3, T1 = T2$$

In this example, each view and the citation query pair, (V_i, C_{v_i}) where $i = 1, 2, \ldots, 5$ composes a citation view. Note that, V_1 is parameterized by the primary key G, which indicates that each view tuple in the instance of V_1 should carry unique set of contributor

names. This is achieved by using the citation query C_{V_1}, which shares the same parameters as V_1 and aggregates the contributor names by the attribute G (the function Array_agg is used to collect all contributor names as a list), which ends up with one aggregated list of contributor names for each individual tuple in Gene.

Table 3.7 – Instance of view V_1

	G	Ty
$t_{v_1}1$	1	TEC
$t_{v_1}2$	2	rRNA

Table 3.8 – Instance of view V_2

	Type	COUNT(G)
$t_{v_2}1$	rRNA	3

Table 3.9 – Extended instance of view V_1

	G	N	Ty
$t_{v_{e1}}1$	1	TF	TEC
$t_{v_{e1}}2$	2	FH	rRNA

Table 3.10 – Extended Instance of view V_2

	GID	Name	Type
$t_{v_{e2}}1$	2	FH	rRNA
$t_{v_{e2}}2$	3	RP1	rRNA
$t_{v_{e2}}3$	4	IYD	rRNA

3.1.2 View mappings and Query extensions

Based on the above *citation view model*, I found that the data citation problem is closely related to the classical query rewriting using views problem. However, unlike the query rewriting problem where only the schema analysis between view schema and query schema is required, it is also essential to reason about the finer-grained dependency between each individual input *view tuples* and each individual *query tuples*, which means that a qualified view may not necessarily rewrite the user query at the schema level, but may still include tuples which can contribute to some query tuple with some specific database instance. Same as the query rewriting using views problem, I reason about the qualified view tuples for each individual query tuple under the *view mappings*, which map the variables and atoms used

in the view schema to the query schema. The definition of *view mappings* is provided as below, which is borrowed from the classical query rewriting using views problem:

Definition 2. View Mapping *Given a view definition* $\mathtt{V}$ *and query* $\mathtt{Q}$

$$\mathtt{V}(\bar{\mathtt{Y}}) : -\mathtt{A}_1(\bar{\mathtt{Y}_1}), \mathtt{A}_2(\bar{\mathtt{Y}_2}), \ldots, \mathtt{A}_k(\bar{\mathtt{Y}_k}), \mathtt{condition}(\mathtt{V})$$
$$\mathtt{Q}(\bar{\mathtt{X}}) : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_m(\bar{\mathtt{X}_m}), \mathtt{condition}(\mathtt{Q})$$

a **view mapping** M *from* V *to* Q *is a tuple* (h, ϕ) *in which:*

- h *is a partial one-to-one function which 1) maps a relational subgoal* A_i *in* V *to a relational subgoal* B_j *in* Q *with the same relation name; and 2) cannot be extended to include more subgoals of* Q*.*
- ϕ *are the variable mappings from* $\bar{Y}' = \cup_{i=1}^{k} \bar{Y_i}$ *to* $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$ *induced by* h

A relational subgoal B_j *of* Q *is* covered *iff* $h(A_i) = B_j$ *for some* i*. A variable* x *of* Q *is* covered *iff* $\phi(y) = x$ *for some* y*.*

As described above, *view mappings* construct mappings of all variables between query body and view body, which thus motivates us to introduce the extended version of query schema by including all variables in the query body in the query head. i.e.:

Definition 3. Query Extension *Given a query*

$$\mathtt{Q}(\bar{\mathtt{X}}) : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_m(\bar{\mathtt{X}_m}), \mathtt{condition}(\mathtt{Q})$$

where $condition(Q)$ *are the non-relational subgoals, the* extension of Q*,* Q_{ext}*, is*

$$\mathtt{Q}_{\mathtt{ext}}(\bar{\mathtt{X}}') : -\mathtt{B}_1(\bar{\mathtt{X}_1}), \mathtt{B}_2(\bar{\mathtt{X}_2}), \ldots, \mathtt{B}_m(\bar{\mathtt{X}_m}), \mathtt{condition}(\mathtt{Q})$$

where $\bar{X}' = \cup_{i=1}^{m} \bar{X}_i$*. Note that* $\bar{X} \subseteq \bar{X}'$*.*

Example 2. *Consider the following query which finds the pairs of gene IDs and transcript name from all joint tuples between all 'rRNA' genes and transcripts:*

$$Q_1(GID1, Name2) : -Gene(GID1, Name1, Type1)$$
$$, Transcript(TID, Name2, Type2, GID2)$$
$$, Type2 = \text{`}rRNA\text{'}, GID1 = GID2$$

The query instance of Q_1 *is provided in Table 3.11. According to Definition 3, the extension of* Q_1 *is:*

$$Q_{ext1}(GID1, Name1, Type1, TID, Name2, Type2, GID2)$$
$$: -Gene(GID1, Name1, Type1)$$
$$, Transcript(TID, Name2, Type2, GID2)$$
$$, Type2 = \text{`}rRNA\text{'}, GID1 = GID2$$

There are obvious view mappings from V_1, V_2, V_7 *and* V_8 *to the body of* Q*. For example, one possible mapping,* M_{11}*, maps the first (and only) subgoal of* V_1 *to the first subgoal of* Q_1 *and induces the mapping of variables* $\phi(G) = GID1,\ \phi(N) = Name1,\ \phi(Ty) = Type1$*. Another mapping,* M_{51}*, maps the first and second subgoal of* V_5 *to the first and second subgoal of* Q_1 *respectively and induces the mapping of variables* $\phi(G) = GID1,\ \phi(N) = Name1,\ \phi(Ty1) = Type1,\ \phi(T) = TID,\ \phi(N') = Name2,\ \phi(Ty2) = Type2,\ \phi(G') = GID2$*.*

By going through all the views, all the possible view mappings for Q_1 *can be derived, which are presented in Table 3.13.*

Table 3.11 – Instance of Q_1

	GID1	Name2
$t_{q_1}1$	2	PC-203
$t_{q_1}2$	2	HP-218
$t_{q_1}3$	3	TP-208

Table 3.12 – Extended instance of Q_1

	GID1	Name1	Type1	TID	Name2	Type2	GID2
$t_{q_{e1}}1$	2	FH	rRNA	2	PC-203	rRNA	2
$t_{q_{e1}}2$	2	FH	rRNA	4	HP-218	rRNA	2
$t_{q_{e1}}3$	3	RP1	rRNA	5	TP-208	rRNA	3

Table 3.13 – All possible view mappings for $Q1$

View	View mapping	h: mappings on relations	ϕ: mappings on variables	Subgoals covered
V_1	M_{11}	gene → gene	G → GID, N → Name1 , Ty → Type1	gene
V_4	M_{41}	Transcript → Transcript	T → TID, N → Name2 , Ty → Type2, G → GID2	Transcript
V_5	M_{51}	Transcript → Transcript , gene → gene	G → GID, N → Name1 , Ty → Type1, T → TID , N′ → Name2, Ty2 → Type2 , G′ → GID2	gene, Transcript

Then based on the definition of the view mappings, the positional mappings between individual attributes in view schema and individual attributes in query schema can be derived, under which I can map a view tuple t_v to the query instance and thus the existence of the mapped view tuple in the query instance determines whether t_v contributes to the construction of certain query tuples or not. For example, under the view mapping M_{11}, the attributes of V_1, i.e., G, N, Ty are mapped to $GID1, Name1$ and $Ty1$ respectively, under which if the view tuple $t_{v_{c1}2}$=(2,'FH','rRNA') with $G = 2$, $N =$'FH' and $Ty =$'rRNA' appears in the query instance, then there should exist one query tuple with $GID1 = 2$,$Name1 =$'FH' and $Ty1 =$'rRNA' in the extended instance of Q_1, which is true for both $t_{q_{e1}1}$ and $t_{q_{e1}2}$ as shown in Table 3.12. This indicates that $t_{v_{c1}2}$, i.e. $t_{v_1 2}$ after the duplicates are removed, is responsible for the contributions of the query tuple $t_{q_1 1}$ and $t_{q_1 2}$. Therefore, the key step to produce fine-grained citations to each query tuple is to identify whether a view tuple can be transformed to some portion of a query tuple under certain view mapping, which is equivalent to determining the validity of the view mappings for a query tuple. In what follows, I will introduce how to determine the validity of view mappings at query tuple level with two different types of approaches, i.e. Rewriting-Based Approach and Provenance-Based Approach.

3.2 Reasoning about validity of view mappings

I propose two types of models, Rewriting-Based Approach and Provenance-Based Approach in [12] and [13] respectively to determine whether a view tuple can contribute to the con-

struction of a certain query tuple. The former approach extends query rewriting using views to the tuple-level to analyze which combinations of view tuples could "rewrite" a certain query tuple, which is applicable for general SPJ queries and SPJ views but fails to provide correct results for aggregate queries and aggregate views as [13] shows. This thus motivates us to come up with the other approach, i.e. Provenance-Based Approach, which deals with the data citation problem by explicitly employing the provenance-semiring model. The details of the two approaches are provided in the next subsection.

Note that a view may provide more than one view mappings for a given query. For example, given a query computing all pairs of gene IDs for the gene type 'rRNA', which needs to calculate a cross-product on two copies of 'rRNA' 'Gene' tuples, i.e.:

$$Q_2(G1, G2) : -Gene(G1, N1, Ty1), Gene(G2, N2, Ty2), Ty1 = `rRNA', Ty2 = `rRNA'.$$

This query body includes two copies of Gene relation, thus producing two view mappings M_{21} and M'_{21} mapping V_1 to Q_2 according to Definition 2, each of which maps the gene relation in the body of V_1 to one gene relation in the body of Q_2. As a result, the validity of the two view mappings for each query tuple is analyzed individually.

3.2.1 Rewriting-Based Approach (RBA)

Based on the intuition above, I illustrated how to determine the validity of view mappings for some query tuple with Rewriting-Based Approach (RBA), in which the provenance is *implicitly* used and I only consider the view mappings that map *conjunctive views* to *conjunctive queries*:

Definition 4. Valid View Mapping *By reusing the notations from Definition 2, given a database instance D, a view mapping $M = (h, \phi)$ of a conjunctive view V is* valid *for a tuple $t \in Q_{ext}(D)$ iff:*

- *The projection of t on the variables that are mapped in Q_{ext} under the mapping ϕ is a tuple in $V_{ext}(D)$: $\Pi_{\phi(\bar{Y}')}t \in V_{ext}(D)$ (recall that in Defi'nition 2, $\bar{Y}' = \bigcup_{i=1}^{k} \bar{Y}_i$,*

representing a set of all the variables from all the subgoals in the view body)

- *There exists at least one variable $y \in \bar{Y}$ such that $\phi(y)$ is a distinguished variable (recall that $\bar{Y}$ represents a set of all the head variables of the view V)*
- *All lambda variables in V are mapped to variables in $\bar{X}'$.*

Example 3. *Let us revisit the intuitive example presented above. Recall that under the view mapping $M_{11} = (h_{11}, \phi_{11})$, $\bar{Y}' = \{G, N, Ty\}$ and $\bar{Y} = \{G, Ty\}$, the former of which is mapped to the attribute set $\{GID1, Name1, Type1\}$ in the query Q_1. This indicates that $\phi_{11}(\{G, N, Ty\}) = \{GID1, Name1, Type1\}$. In order to determine the validity of view mapping M_{11} for the tuple $t_{q_e1}1$=(2,'FH','rRNA',2,'PC-203','rRNA',2) from the extended instance of Q_1, I project this tuple on the attributes from Q_1 involved in the mapping M_{11}, i.e. $\Pi_{\phi_{11}(\bar{Y}')}t_{q_e1}1 = \Pi_{\phi_{11}(GID2,Name1,Type1)}t_{q_e1}1$ =(2, 'FH', 'rRNA'), which is exactly $t_{v_e1}2$ in the extended instance of V_1 (see Table 3.9). Therefore, the first condition holds. Plus, the last two conditions can be verified since the head variable G (lambda variable also) of V_1 is mapped to $GID1$ under M_{11} which is also the head variable of Q_1. As a consequence, M_{11} is a valid view mapping for the query tuple $t_{q_e1}1$.*

As shown in Definition 4, the core idea of checking the validity of a given view mapping for a query tuple t is to evaluate whether the portion of t projected on the attributes involved in the view mappings, t_p should appear in the extended instance of the view or not, which is equivalent to check the satisfiability of the conditions appearing in the view body for t_p. For example, it has been demonstrated that in Example 3, the projected portion t_p =(2, 'FH', 'rRNA') for the query tuple $t_{q_e1}1$ appear in the extended instance of V_1, which also satisfies the condition $G \leq 2$ in the body of view V_1. In contrast, for the query tuple $t_{q_e1}3$, by projecting on the same set of attributes as $t_{q_e1}1$, the resulting tuple (3, 'RP1', 'rRNA') is missing from the extended instance of V_1 due to the violation of the condition $G \leq 2$.

The intuitive example above reveals the necessity of reasoning at fine-grained level, i.e. tuple-level, which thus becomes computationally challenging especially for large query instances. To deal with this, I proposed two different ways to effectively identify the valid

view mappings for each query tuple, i.e. Tuple-level approach (TLA) and semi-schema-level approach (SSLA), which can effectively check the satisfiability of the conditions for each query tuple under the view mappings. The main difference between the two approaches is how to tackle the *local predicates*, which are the conditions in the query body or view body only involving attributes from one base relation, e.g. $G < 2$ in the body of view V_1. For the conditions with attributes from different base relations, e.g. $GID1 = GID2$ in the body of the query Q_1, they are defined as *global predicates*. In what follows, TLA and SSLA are described respectively.

3.2.1.1 Tuple-level approach (TLA)

The *tuple-level approach* is composed of the *preprocessing step*, the *Query execution step* and the *reasoning step*, which generates valid view mappings in the end, preparing for the follow-up citation generation phase. To begin with, to facilitate checking local predicates in TLA, the database schema is modified: A *view vector* column is added to each relation identifying all views in which a tuple potentially participates. For each view $V :- B_V$, V is added to the view vector of each tuple t in relation $R \in B_V$ whenever t satisfies the *local predicates* for V. This reduces the overhead for checking the *local predicates* at query time, and filters out invalid view mappings early. Any *global predicates* are checked at query time.

Example 4. *Let us revisit Example 1 and consider annotating all conjunctive views (i.e. V_1, V_4, V_5, V_6) to the base relations Gene, Transcript and Exon. The annotated base relations are presented in Table 3.14-3.16.*

Since there is one local predicate in V_1 which filters out all the gene tuples with gene ID larger than 2, then only the first two gene tuples are annotated with view V_1 as Table 3.15. Similarly, the existence of the local predicates $T \geq 2$ and $E \leq 2$ in the view V_4 and V_6 respectively lead to the annotations of V_4 and V_6 to the last three Transcript tuples and the first two exon tuples respectively as Table 3.16 and Table 3.14 show. For V_5 which is a multi-relation view, it includes a local predicate $T \geq 4$ and a global predicate $G = G'$, the former of which influences the annotations of V_5 to the relation Transcript. Since there

Table 3.14 – Instance of relation *Exon* with annotated candidate views

	EID	Level	TID	View_vector
t_{e1}	1	1	1	V_6
t_{e2}	2	3	2	V_6
t_{e3}	3	2	2	
t_{e4}	4	2	2	

Table 3.15 – Instance of relation *Gene* with annotated views

	GID	Name	Type	View_vector
t_{g1}	1	TF	TEC	V_1, V_5
t_{g2}	2	FH	rRNA	V_1, V_5
t_{g3}	3	RP1	rRNA	V_5
t_{g4}	4	IYD	rRNA	V_5
t_{g5}	5	EPN	mRNA	V_5

Table 3.16 – Instance of relation *Transcript* with annotated views

	TID	Name	Type	GID	View_vector
t_{t1}	1	MB-203	TEC	1	V_6
t_{t2}	2	PC-203	rRNA	2	V_4, V_6
t_{t3}	4	HP-218	rRNA	2	V_4, V_5, V_6
t_{t4}	5	TP-208	rRNA	3	V_4, V_5, V_6

exists no local predicates for the relation Gene which is also in the definition of V_5*, then all the tuples in the relation Gene are annotated with* V_5 *as Table 3.15 shows. The reasoning of the satisability of the global predicate* $G = G'$ *in* V_5 *is left for later phases.*

Preprocessing step. When a query $Q : -B_Q$ is submitted, I first calculate all *possible* view mappings using the view and query schemas. Some of these mappings may become invalid for individual result tuples depending on whether global predicates for the views hold. In order to enable global predicate checking as well as the evaluation of parameterized views, Q is then extended to include: 1) lambda variables under all possible view mappings (which are used to evaluate parameterized views); 2) view vectors of every base relation occurring in B_Q; and 3) columns representing the truth value of every global predicate under every possible view mapping (which are used to filter out invalid view mappings based on global predicates). In the three extra types of columns in the extended query schema, the latter two are used to evaluate the validity of view mappings while the first one is for later use of extracting citation information (recall that according to the definition of citation views,

i.e. Definition 1, view tuples with different values for lambda variables may have different citations). The details of the *preprocessing step* are presented in Algorithm 1, which takes as input a set of views $\mathcal{V}$ and a user query Q to generate the extended query Q_{ext1} and all possible view mappings.

Algorithm 1: Preprocessing step

Input : a set of views: $\mathcal{V} = \{V_1, V_2, ..., V_k\}$, user query: $Q(\bar{X}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$

Output: the set of all possible view mapping $\mathcal{M}$, the extended query $Q_{ext1}(\bar{X'})$

Initialize $\mathcal{M} = \{\}$
Initialize the schema of the extended query $\bar{X'} = \bar{X}$
for *each view* $V \in \mathcal{V}$ **do**
    Derive all possible view mappings from V to Q that follows definition 2 and the last two conditions in definition 4 and add them to $\mathcal{M}$.
end
for *each view mapping* $M \in \mathcal{M}$ **do**
    Derive lambda terms $L(M)$ and the predicates $condition(M)$ under M
    Add all lambda terms in $L(M)$ to $\bar{X'}$
    Add boolean expressions of all the global conditions in $condition(M)$ to $\bar{X''}$
end
for *each relation* B_i *in the body of* Q **do**
    Add the view vectors $Vec(B_i)$ to $\bar{X'}$.
end
Construct the extended query Q_{ext1} with the following form:
$Q_{ext1}(\bar{X'}) : -B_1(\bar{X_1}), B_2(\bar{X_2}), \ldots, B_m(\bar{X_m}), condition(Q)$
return $\mathcal{M}$, $Q_{ext1}(\bar{X'})$

Example 5. *Continuing Example 2, I know that there are three view mappings* M_{11}, M_{14} *and* M_{15} *mapping* V_1, V_4 *and* V_5 *to* Q_1 *respectively as Table 3.13 indicates, in which* V_1 *has one lambda variable* G *and* V_5 *has one global predicate* $G = G'$ *and one lambda variable* $Ty2$*. Under the view mapping* M_{11} *and* M_{51}*, the lambda variable* G *and the global predicate* $G = G'$ *are transformed to* $GID1$ *and* $GID1 = GID2$ *respectively, which are included in the extended schema of* Q_1 *along with the view vector columns from the base relations included in* Q_1*, i.e.:*

$$
\begin{aligned}
Q_{r1}(&GID1, Name2, GID1, Type2, (GID1 = GID2), Gene.view\_vector \\
&, Transcript.view\_vector) \\
&: -Gene(GID1, Name1, Type1), Transcript(TID, Name2, Type2, GID2), \\
&Type2 = \text{`}rRNA\text{'}, GID1 = GID2
\end{aligned}
$$

In this extended query schema, I use Q_{r1} to denote the extended query for Q_1 and the expression $(GID1 = GID2)$ to explicitly evaluate the predicate $GID1 = GID2$ in the query instance. Note that the attribute $GID1$ also appears as the head variable in Q_1. Therefore, only one copy of this attribute in the extended query schema can be removed, which can be further simplified by removing $(GID1 = GID2)$ and thus avoiding evaluating this predicate during the query execution phase since $GID1 = GID2$ is the global predicate of Q_1, implying the satisfiability of this predicate for all query tuples. In the end, the simplified extended query definition becomes:

$$Q_{r1}(GID1, Name2, Type2, Gene.view\_vector, Transcript.view\_vector)$$
$$:- Gene(GID1, Name1, Type1), Transcript(TID, Name2, Type2, GID2),$$
$$Type2 = \text{`}rRNA\text{'}, GID1 = GID2$$

Query execution step. The extended query is then executed over the database instance D, yielding an instance over which the reasoning of valid view mappings can occur in the next step.

Example 6. *Continuing Example 5, Q_{r1} is executed against the database instance shown in Table 3.1-3.6, resulting in the following query instance:*

Table 3.17 – Instance of Q_{r1}

	GID1	Name2	Type2	Gene.view_vector	Transcript.view_vector
$t_{q_{r1}1}$	2	PC-203	rRNA	V_1, V_5	V_4, V_6
$t_{q_{r1}2}$	2	HP-218	rRNA	V_1, V_5	V_4, V_5, V_6
$t_{q_{r1}3}$	3	TP-208	rRNA	V_5	V_4, V_5, V_6

Reasoning step. In this step, the major concern is to check the satisfiability of the *global predicates* from the multi-relation views for each individual query tuple. A view mapping M for a multi-relation view V is valid for a query tuple t iff all *global predicates* under this mapping are true for t and the annotation of V appears in *all* the relations involving in the mapping M. Invalid view mappings are then removed from the view vectors. The details of this step are shown in Algorithm 2, which takes as input the database instance

D, the output from Algorithm 1 (i.e. a set of view mappings and the extended query Q_{ext1}) and a query tuple t from the query instance obtained through the *Query execution step* to construct valid view mappings for each query tuple t.

Example 7. *Continuing Example 6, the validity of the view mappings for the single-relation views depends on the occurrence of their annotations in the corresponding view vector columns. For example,* V_1 *appears in the Gene's view vector for the first query tuples, indicating the satisfiability of the local predicates involving Gene relation for those tuples, which has been evaluated early when those views are annotated to the base relation tuples.*

In contrast, for multi-relation views, both their view annotations and the evaluation results of global predicates *in the resulting query instance are essential to determine the validity of their view mappings* V*, Since there are no global predicates explicitly evaluated during the query execution phase of* Q_{r1}*, then the occurrence of the view annotations is crucial to determine the validity of view mappings. For example,* V_5 *joins Gene relation and Transcript relation, which appears in both of the Gene's view vector column and Transcript's view vector column for the last two query tuples in the instance of* Q_{r1}*, implying its satisfiability of the local predicates involving both Gene relation and Transcript relation for those query tuples. Therefore, the corresponding view mapping of* V_5*, i.e.* M_{51} *should be a valid view mapping for each query tuple. In contrast,* V_5 *does not appear in the Transcript's view vector in the tuple* $t_{q_{r1}1}$*, indicating its violations of the local predicates in* V_5 *involving the Transcript relation, i.e.* $T \geq 4$ *from* V_5*. Furthermore, despite the existence of* V_6 *in the Transcript's view vectors, it cannot provide valid view mappings for any query tuples in Table 3.17. This is because another relation appearing in* V_6*, i.e. Exon is missing from the body of* Q_{r1}*, thus failing to construct view mappings from* V_6 *to* Q_{r1}*. In the end, the valid view mappings for each individual query tuple are listed in Table 3.18.*

Table 3.18 – Instance of Q_{r1} with valid view mappings

	GID1	Name2	Valid view mappings
$t_{q_{r1}1}$	2	PC-203	M_{11}, M_{41}
$t_{q_{r1}2}$	2	HP-218	M_{11}, M_{41}
$t_{q_{r1}3}$	3	TP-208	M_{41}, M_{51}

Algorithm 2: Determine the valid view mappings in TLA

Input : Database instance D, the set of all the possible view mappings $\mathcal{M}$, the schema of the extended query Q_{ext1}: $(\bar{X}')$, and a tuple $t \in Q_{ext1}(D)$

Output: A set of valid view mapping sets $\mathcal{M}(t)$, a set of maximally covered relations $MCR(t)$

Initialize $\mathcal{M}(t) = \{\}$

`/* M(t) denotes a set of valid view mapping */`

for *each view vector* $Vec(B_i)(i = 1, 2, \ldots, m)$ **do**

  for *each annotated view* V *in* $Vec(B_i)$ **do**

    for *each view mapping* M *that the view* V *is involved in* **do**

      check whether each view mapping M satisfies:

        1. the first condition in the definition 4 by checking whether all of the boolean expressions of the global predicates in $condition(M)$ are true and
        2. B_i is covered by the mapping M and
        3. if M covers more than one relations, V should appear in every view vector of those relations.

      if *M follows all the three rules above* **then**

        add M to $\mathcal{M}(t)$

        add all the relations in Q that M covers into $MCR(t)$

      end

    end

  end

end

return $\mathcal{M}(t)$ and $MCR(t)$

3.2.1.2 Semi-Schema-level approach (SSLA)

Unlike TLA, there is no need to annotate the base relations with views with Semi-Schema-level approach (SSLA). Instead, both the local predicates and the global predicates are evaluated during the executions of the extended queries. The algorithmic details are presented as below step by step.

Preprocessing step As before, when a user query $Q :- B_Q$ is submitted, all the possible view mappings are calculated. The query is extended to include 1) lambda variables under all the possible view mappings; and 2) columns representing the truth value of every global and local predicate. Since base relations are not annotated, no view vectors are returned. The extended query is then executed on the database yielding an extended query instance used for reasoning valid view mappings.

Example 8. *Similar to Example 5, I know that there are three view mappings* M_{11}, M_{14}

and M_{15} mapping V_1, V_4 and V_5 to Q_1 respectively as Table 3.13 indicates. To construct the extended query Q'_{r1}, all the lambda variables (i.e. the lambda variable G from V_1), local predicates (i.e. the local predicates $G \leq 2$ from V_1, $T \geq 2$ from V_3 and $T \geq 4$ from V_5) and global predicates (i.e. $G = G'$ from V_5) of those views under the corresponding view mappings are included in the query schema V_1, i.e.:

$$Q'_{r1}(GID1, Name2, Type2, GID1, (GID1 = GID2), (G \leq 2), (TID \geq 2), (TID \geq 4))$$
$$:- Gene(GID1, Name1, Type1), Transcript(TID, Name2, Type2, GID2),$$
$$Type2 = \text{'rRNA'}, GID1 = GID2$$

Again, $(GID1 = GID2)$ is removed since it exists in the query body and one copy of $GID1$ are removed due to the duplicates, which results in the following simplified Q'_{r1}:

$$Q'_{r1}(GID1, Name2, Type2, (G \leq 2), (TID \geq 2), (TID \geq 4))$$
$$:- Gene(GID1, Name1, Type1), Transcript(TID, Name2, Type2, GID2),$$
$$Type2 = \text{'rRNA'}, GID1 = GID2$$

After executing this extended query on the database instance shown in Table 3.1-3.6, the query results are presented in Table 3.19:

Table 3.19 – Instance of Q'_{r1}

	GID1	Name2	Type2	$(G \leq 2)$	$(TID \geq 2)$	$(TID \geq 4)$
$t_{q_{r1'}1}$	2	PC-203	rRNA	True	True	False
$t_{q_{r1'}2}$	2	HP-218	rRNA	True	True	True
$t_{q_{r1'}3}$	3	TP-208	rRNA	False	True	True

Reasoning step In this step, the set of valid view mappings for each tuple $t \in Q_{ext2}(D)$, $VM(t)$, is derived based on the truth values of the *global* and *local predicates* (all must be true for a view mapping to be in $VM(t)$).

Example 9. *Continuing Example 8, it is not hard to know that the local predicates $(G \leq 2)$, $TID \geq 2$ and $TID \geq 4$ are the only local predicates in V_1, V_3 and V_5 respectively, which correspond to view mappings M_{11}, M_{31} and M_{51} respectively. Therefore, the truth values*

Algorithm 3: Determine the valid view mappings in SSLA

Input : Database instance D, The set of all the possible view mappings $\mathcal{M}$, the extended query: $Q_{ext2}(\bar{X}') : -B_1(\bar{X}_1), B_2(\bar{X}_2), \ldots, B_m(\bar{X_m}), condition(Q)$, and a tuple $t \in Q_{ext2}(D)$

Output: A set of valid view mappings $\mathcal{M}(t)$, a set of maximally covered relations $MCR(t)$

```
Initialize M(t) = {}
Initialize MCR(t) = {}
for each view mapping M in M do
    check whether M satisfies the first condition in the definition 4 by checking whether all of the
      boolean expressions of condition(M) are true
    if M follows the rule above then
        add M to M(t)
    end
    add all the relations in Q that M covers into MCR(t)
end
return M(t) and MCR(t)
```

of those predicates in the query tuples in Table 3.19 which reflect the satisfiability of those predicates should imply the validity of those view mappings. For example, for the first query tuple $t_{q'_{r1}1}$, the evaluation results for the predicates $(G \leq 2)$, $TID \geq 2$ and $TID \geq 4$ are True, True and False respectively, meaning that only the former two predicates hold for this query tuple, thus justifying the validity of the corresponding view mappings M_{11} and M_{31} for this query tuple. In the end, it ends up with the same set of valid view mappings for each query tuple as the one in Table 3.18.

3.2.1.3 Optimization in the implementations

Deriving valid view mappings tuple by tuple is time-consuming especially when the query result is very large. However, it is possible to find subsets of tuples that will share the same set of valid view mappings using the view vectors and boolean values of local and global predicates returned in the extended query, by which deriving valid view mappings can be done once per group and then propagated to all tuples within the group. For example, in Table 3.18, the first two tuples form one group and the third and fourth tuples form another group. This optimization leads to significant performance gains.

3.2.2 Provenance-Based Approach (PBA)

Although Rewriting-Based Approach can provide perfect solutions for the scenarios where only conjunctive views and conjunctive views exist, it overlooked the widely use of aggregate queries and aggregate views in scientific databases.

One such example is Hetionet, a database that "encodes" biology by integrating various types of biological information from different publicly available resources [119, 64]. As data is copied from these source datasets, citation information (generally in the form of traditional publication IDs) is also copied and should be propagated to the results of queries. The majority of queries against this database involve aggregation to retrieve statistical information.

Another example, which requires both aggregate queries and aggregate views, is GENCODE [63], an encyclopedia of genes and gene variants whose goal is to identify all functional elements in the human genome using annotations. The gene annotation process involves a combination of automatic annotation, manual annotation, and experimental validation. For genes that are manually annotated, information is maintained about the responsible research groups. Statistics are also provided for every gene – an *aggregate view* over the genes – which has another type of citation giving credit to the creators of the aggregate view. Common queries over GENCODE also involve aggregation. For instance, one query computes statistics for every *type* of gene.

To deal with aggregate queries and aggregate views in the context of data citation application, one natural idea is to leverage the classical query rewriting using views with aggregation algorithms, e.g. [78, 79]. However, as revealed in the next subsection, simply adjusting query rewriting using views with aggregation algorithms to Data citation application is not enough, sometimes resulting in unreasonable results, thus justifying the use of provenance.

3.2.2.1 The need for provenance

Example 10. *Consider the following query:*

$Q_3(T, COUNT(Gid)) : -Gene(Gid, Name, T), Gid <= 3$

and the same query without aggregation:

$Q_4(T, Gid) : -Gene(Gid, Name, T), Gid <= 3$

Our goal is to determine the valid view mappings for the aggregate query Q_3*. Note that only* V_1 *and* V_2 *can provide candidate view mappings* $M_{13}(= (h_{13}, \phi_{13}))$ *and* $M_{23}(= (h_{23}, \phi_{23}))$ *for* Q_3 *since they include the same relational subgoals in the body.* M_{13} *and* M_{23} *have the same form, i.e,* $h_{13}(h_{23}) = Gene(G,\ N, Ty) \rightarrow Gene(Gid, Name, T)$ *and* $\phi_{13}(\phi_{23}) = \{G \rightarrow Gid, N \rightarrow Name, Ty \rightarrow T\}$*. Under the two mappings, neither* V_1 *nor* V_2 *can be used to rewrite* Q_3 *for an arbitrary database instance since* V_1 *has one logically stronger predicate than* Q_3*, and there exists an instance* D *for* V_2 *such that the first three genes are not 'rRNA'. In this case,* V_2 *and* Q_3 *aggregate over different set of tuples from the Gene relation.*

Table 3.20 – Instance of relation $Exon$ with provenance

	EID	Level	TID	prov
t_{e1}	1	1	1	e_1
t_{e2}	2	3	2	e_2
t_{e3}	3	2	2	e_3
t_{e4}	4	2	2	e_4

Table 3.21 – Instance of relation $Gene$ with provenance

	GID	Name	Type	prov
t_{g1}	1	TF	TEC	g_1
t_{g2}	2	FH	rRNA	g_2
t_{g3}	3	RP1	rRNA	g_3
t_{g4}	4	IYD	rRNA	g_4
t_{g5}	5	EPN	mRNA	g_5

Table 3.22 – Instance of relation $Transcript$ with provenance

	TID	Name	Type	GID	prov
t_{t1}	1	MB-203	TEC	1	r_1
t_{t2}	2	PC-203	rRNA	2	r_2
t_{t3}	4	HP-218	rRNA	2	r_3
t_{t4}	5	TP-208	rRNA	3	r_4

Still, some *query tuples may be computed using view tuples of* V_1 *and* V_2 *given a database instance* D*, which captures the concept of* fine-grained citation *proposed in [12]. To illustrate,*

I use the instance provided in Tables 3.1-3.6, which result in the instances of V_1, V_2 and Q_3 shown in Table 3.7, Table 3.8 and Table 3.23 (ignore the last three columns for now).

Table 3.23 – Instance of Q_3 with how-provenance

	T	COUNT(Gid)	$agg((G \leq 2))$	$agg((Ty = \text{`}rRNA\text{'}))$	prov
$t_{q_3}1$	TEC	1	{T}	{F}	g_1
$t_{q_3}2$	rRNA	2	{T, F}	{T, T}	$g_2 + g_3$

Table 3.24 – Instance of Q_4 with how-provenance

	T	Gid	$(G \leq 2)$	$(Ty = \text{`}rRNA\text{'})$	prov
$t_{q_4}1$	TEC	1	T	F	g_1
$t_{q_4}2$	rRNA	2	T	T	g_2
$t_{q_4}3$	rRNA	3	F	T	g_3

To follow the idea of checking the existence *of a query tuple in the view instance, a plausible approach is to extend RBA to aggregate queries by employing a special built-in function agg (the "array_ agg" function in PostgreSQL) to collect the evaluations of the view predicates for every query tuple before aggregation is applied. For example, Q_3 could be extended as:*

$$Q_3'(T, COUNT(Gid), agg((Gid \leq 2)), agg((T = \text{`}rRNA\text{'}))) : -Gene(Gid, Name, T), Gid <= 3$$

which ends up with the instance of Q_3 shown in Table 3.23 along with the evaluation results of view predicates collected by agg (see the third and fourth columns). This can be also regarded as aggregating the results of the following query (denoted by Q_4') by explicitly evaluating the predicates of V_1, V_2 in Q_4:

$$Q_4'(T, Gid, (Gid \leq 2), (T = \text{`}rRNA\text{'})) : -Gene(Gid, Name, T), Gid <= 3$$

which generates the instance shown in Table 3.24. Note that, since two tuples are generated before the aggregate function is applied for tuple $t_{q_3}2$ (i.e. $t_{q_4}2$ and $t_{q_4}3$ from Table 3.24), the evaluation of $agg((Gid \leq 2))$ will result in a set *of boolean values of size 2, which, intuitively, is derived by aggregating over $t_{q_4}2$ and $t_{q_4}3$ in Table 3.24. The existence of "false" indicates that before aggregation, one query tuple (i.e. $t_{q_4}3$) was missing from the view instance $V_1(D)$,*

thus V_1 is not valid for $t_{q_3 2}$. In contrast, for $t_{q_3 1}$ the evaluation of $agg((Gid \leq 2))$ is only "true", meaning that V_1 is valid.

However, simply checking the existence of query tuples in the view instance for aggregate views is not enough. *For example, for query tuple $t_{q_3 2}$, although the evaluations of the predicate Ty = '$rRNA$' (from V_2) does not include "false", indicating that all the query tuples before aggregation exist in the view instance, the aggregate result of $t_{q_3 2}$ (i.e. 2) does not match that of $t_{v_2 1}$ (i.e. 3). The reason is that three tuples $t_{g2} - t_{g4}$ from relation Gene are used to construct $t_{v_2 1}$, while $t_{q_3 2}$ is derived from only two of them ($t_{g2} - t_{g3}$). This means that the reasoning should not only capture the* existence *of query tuples in the view instance but also the* exact matching *of the aggregate results between query tuples and view tuples. I therefore adopt an alternative model, called the* Provenance-Based Approach *(*PBA*), which uses* provenance *and is described in the following subsection.*

3.2.2.2 Preliminary of PBA

To begin with, I introduce some key concepts, ready for introducing the details of PBA.

Granularity of queries and views. An essential step in determining the validity of a view mapping $M = (h, \phi)$ is to compare the schemas of Q and V, and detect whether V keeps all necessary variables in its head. In particular, if the aggregate view V has the set of grouping variables $\{Y_1, Y_2, \ldots, Y_m\}$, then $\{\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_m)\}$ should be a superset of the grouping variables of Q, $\{X_1, X_2, \ldots, X_k\}$. If $\{\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_m)\} = \{X_1, X_2, \ldots, X_k\}$, then Q has the *same granularity* as V. Otherwise, if $\{\phi(Y_1), \phi(Y_2), \ldots, \phi(Y_m)\} \supsetneq \{X_1, X_2, \ldots, X_k\}$, then V has *finer granularity* than Q.

Table 3.25 – $Q_5(D)$ with how-provenance

	G	G'	prov
$t_{q_5 1}$	2	2	$r_2 * r_2 + r_2 * r_3 + r_3 * r_2 + r_3 * r_3$

How-provenance normal form I use the notion of *how-provenance* introduced in [7]. To simplify reasoning over how-provenance polynomials, [120] defines a normal form as follows: first, the provenance tokens in each how-provenance monomial preserves the same

order as the relational subgoals in the query body. Second, the exponent of every provenance token is forced to be 1. Third, the coefficient of every monomial in a how-provenance polynomial is forced to be 1 by breaking the monomials with coefficient greater than 1 into multiple how-provenance monomials, each corresponding to an *assignment* of the query atoms to database tuples.

Example 11. *Consider the following query:*

$$Q_5(G, G') : -Transcript(T, N, Ty, G), Transcript(T', N', Ty', G'), T >= 2, Ty = Ty' \\ , T' >= 2$$

The provenance-aware query result is shown in Table 3.25 using the instance of Transcript *in Table 3.5. Note that the second and the third monomials for the tuple $t_{q_5 1}$ are equivalent but correspond to different assignments, hence written differently ($r_2 * r_3$ vs $r_3 * r_2$). Further, the first how-provenance monomial of $t_{q_5 1}$ is written as $r_2 * r_2$ instead of the compact form (r_2^2). Furthermore, the coefficient of all the monomials in the how-provenance polynomial of $t_{q_5 1}$ is 1 rather than grouping them together.*

For a query and a result tuple, [120] defines an *isomorphism* between *assignments* and the how-provenance monomials in a query. Borrowing these ideas, I define an isomorphism between relational subgoals and how-provenance monomials under an assignment γ, which relies on the normal form of how-provenance monomials mentioned previously.

Definition 5. Isomorphism between how-provenance monomials and subgoals. *Given a conjunctive or aggregate query Q with relational subgoals $B_1, B_2, \ldots, B_m$, under an assignment γ, base relation tuples $t_{b1}, t_{b2}, \ldots, t_{bm}$ are assigned to relational subgoals B_1, $B_2, \ldots, B_m$ respectively to generate an output tuple, which can be written as $\gamma(B_i) = t_{bi} (i = 1, 2, \ldots, m)$ [8]. If tuple t_{bi} is associated with how-provenance token h_{bi}, then I say that under the assignment γ there is an* isomorphism *F between each relational subgoal B_i and each provenance token h_{bi} (call* isomorphism under an assignment *for short thereafter), which can be written as: $F(B_i|\gamma) = h_{bi}$ and $F^{-1}(h_{bi}|\gamma) = B_i$.*

Returning to Example 11, consider the second provenance monomial $r_2 * r_3$ and corresponding assignment γ in query tuple $t_{q_5 1}$ of Table 3.25. Since t_{t2} and t_{t3} are associated with how-provenance tokens r_2 and r_3 respectively, there should be an isomorphism F under γ such that $F(Transcript(T, N, Ty, G)|\gamma) = r_2$ while $F(Transcript(T', N', Ty', G')|\gamma) = r_3$.

3.2.2.3 PBA for conjunctive queries

The validity conditions of view mappings include *schema-level conditions* and a *tuple-level condition*, which are satisfied by a view mapping M iff it is valid for a given query tuple. The validity conditions guarantee the same result as the conditions in [12] (proof omitted).

Definition 6. Schema-level conditions. *A view mapping M from a conjunctive view V to a conjunctive query Q should satisfy the following conditions at the schema level if it is valid for some query tuples:*

1. *There exists at least one head variable $y \in \bar{Y}$ in V such that $\phi(y)$ is a head variable in Q; and*

2. *All lambda variables in V are mapped to variables in the body of Q.*

Now suppose that head variables $Y_1, Y_2, \ldots, Y_r$ from V are mapped to head variables $X_1, X_2, \ldots, X_r$ from Q, which implies that $\phi(Y_i) = X_i$ $(i = 1, 2, \ldots, r)$. Then I say that the head variables $X_i (i = 1, 2, \ldots, r)$ are *covered* under M. Plus, a relational subgoal of Q is *covered* by M iff it is involved in M.

Definition 7. Tuple-level condition. *Let the how-provenance polynomial of $t_q \in Q(D)$ ($t_v \in V(D)$) include a how-provenance monomial W (W') with corresponding assignment γ (γ') and the isomorphism F (F') under γ (γ'.)*

Given a tuple t_q and a view mapping $M = (h, \phi)$ satisfying the schema-level conditions *above, if a tuple t_v can be found such that the following condition holds, then M is valid for the how-provenance monomial W in t_q: For each relational subgoal A_i in the view body that is involved in the view mapping M and mapped to relational subgoal B_j in the query body under M, then $F(B_j|\gamma) = F'(A_i|\gamma')$.*

Furthermore, one can claim that the how-provenance monomial W' of t_v is mapped *to the how-provenance monomial W of t_q under view mapping M.*

Example 12. *Recall that in Example 10, the instance of Q_4 is shown in Table 3.24. It is not hard to demonstrate that V_1 can provide a valid mapping M_{14} for the query tuple $t_{q_4 1}$ and $t_{q_4 2}$: The* schema-level conditions *are satisfied because the head variable Ty and G (which is also the only lambda variable) in V_1 are mapped to the head variable T and Gid in Q_4 respectively.*

The tuple-level condition *also holds for the two query tuples. For example, for $t_{q_4 2}$ (and view tuple $t_{v_1 2}$), for its single monomial, the assignment and isomorphism under the assignment are γ (γ') and F (F') respectively. Since under the view mapping $M_{14} = (h_{14}, \phi_{14})$, $h_{14}(Gene(G, N, Ty)) = Gene(Gid, Name, T)$, and $F'(Gene(G, N, Ty)|\gamma') = g_2 = F(Gene(Gid, Name, T)|\gamma)$, which thus means that M_{14} is a valid view mapping for the how-provenance monomial g_2 for query tuple $t_{q_4 2}$. One can also prove that M_{14} is a valid view mapping for how-provenance monomial g_1 in tuple $t_{q_4 1}$.*

3.2.2.4 PBA for aggregate queries

The validity conditions for view mappings are next extended to handle aggregate queries and views, using the following intuition: *for a query tuple t, if 1) a set of view tuples can be used to compute t by applying some aggregate function(s) and 2) the view tuples and t are constructed by the same multiset of tuples from the base relations (captured by provenance), then the citation information of those view tuples can be used to construct the citation of t.*

I start by introducing requirements on the aggregate functions before formalizing this intuition.

Aggregate function requirements A view mapping M, which maps an aggregate view V to an aggregate query Q, is valid for a query tuple only if the aggregate functions of V and Q satisfy certain requirements; in particular, [82] formalizes the notion of a *well-formed aggregate function.* Loosely speaking, a well-formed aggregate function can be characterized by some initial “mapper" function, followed by a “reduce" function, followed by a “finalize"

function, which I will call a *terminating function*. It is easy to see that some common aggregate functions such as SUM,MIN, MAX, COUNT and AVG are well formed.

For example, the "mapper" function for AVG takes a set of values, $\{d_1, \ldots, d_k\}$, and maps each number d_i to a pair $(d_i, 1)$. The result of the reduce function is still a pair whose first element represents the sum of the d_i's and second element represents the count (k). The "finalize" function divides the first element by the second element. Similarly, SUM maps each d_i to itself and takes the sum of all d_i's in the reduce step; "finalize" is the identity function.

A well-formed function is *invertible* iff its terminating function is invertible. For example, SUM is invertible whereas AVG is not. Invertibility is important for determining the validity of view mappings when the view has a finer granularity than the query, as illustrated below.

Example 13. *Consider the following query:*

$Q_5(COUNT(G)) : -Gene(G, N, Ty), Ty = \text{`}rRNA\text{'}$

By referencing the definition of the citation views in Section 3.1.1, V_2 computes a coarser-grained aggregation result than Q_5 does. Both of them share the same aggregate function $COUNT$, which is invertible. *This means that the sum of the aggregation results in V_2 can be utilized to attain the result of Q_5 under the obvious view mapping M_{25}, which maps the Gene relation in V_2 to the one in Q_5.*

However, if $COUNT$ is replaced with AVG for both Q_5 and V_2, the aggregation result in V_2 will not be useful to compute the aggregation result in Q_5 under M_{25}; the intermediate sum and count from V_2 that were used in the terminating function (divide) cannot be regained to use in the further aggregation for Q_5, since divide is not invertible.

Computation rules. A view may also be usable to compute the aggregation results in the query without sharing the same aggregate function with the query [82]. For example, the result of an AVG function in the query can be computed by dividing the result of SUM by the result of $COUNT$ from the view. In [81], an aggregate function β is said to be

computed from a set of aggregate functions $\alpha_1, \alpha_2, \ldots, \alpha_n$ if there is a function g such that for any multiset of values M: $\beta(M) = g(\alpha_1(M), \alpha_2(M), \ldots, \alpha_n(M))$. It can be also written as a *computation rule*: $\alpha_1, \alpha_2, \ldots, \alpha_n \rightarrow \beta$. For instance, there is a computation rule from SUM and $COUNT$ to AVG, i.e. $SUM, COUNT \rightarrow AVG$.

The authors in [82] and [81] consider aggregate function requirements for potentially valid views to rewrite a query by combining the properties mentioned above, which are adapted below for data citation:

Definition 8. Aggregate function requirements *Suppose a query Q has an aggregate function α, which takes a set of variables X as arguments. If M is* valid *for some query tuples, the aggregate functions in V should satisfy the following conditions under view mapping $M = (h, \phi)$:*

1. *V also has an aggregate function α with arguments Y, and $\phi(Y) = X$ OR there exists some* computation rule *$\beta_1, \beta_2, \ldots, \beta_m \rightarrow \alpha$ and $\beta_1, \beta_2, \ldots, \beta_m$ also appear in the head of V, all of which take same set of variables Y as arguments and $\phi(Y) = X$.*

2. *If V has finer granularity than Q, then the functions α or $\beta_1, \beta_2, \ldots, \beta_m$ must also be invertible.*

In this case, one can claim that the aggregate term $\alpha(X)$ in Q is covered *under view mapping M.*

3.2.2.5 Valid view mappings for aggregate queries

The conditions for valid view mappings for aggregate queries are formally provided, which are still composed of *schema-level conditions* and a *tuple-level condition*.

Definition 9. Schema-level conditions for aggregate queries. *Given an aggregate query Q and a view mapping $M = (h, \phi)$ from view V to Q. The* schema-level conditions *are as follows:*

1. *For grouping variables of Q, the following must hold:*

(a) *If V is a conjunctive view, then for every grouping variable X of Q there is a head variable Y in V such that $\phi(Y) = X$.*

(b) *If V is an aggregate view, then V must have the same or finer granularity than Q under M.*

2. *There exists at least one aggregate term with aggregate function α taking a set of variables X' as arguments in the head of Q such that:*

(a) *If V is a conjunctive view, then there is a set of head variables Y' in V such that $\phi(Y') = X'$.*

(b) *If V is an aggregate view, then Q and V should satisfy the conditions in Definition 8.*

Suppose the schema-level conditions are satisfied for a view mapping M. M is a *valid view mapping* for some query tuples iff the following *tuple-level condition* holds:

Definition 10. Tuple-level condition for aggregate queries. *Let $t \in Q(D)$ with how-provenance polynomial W. Furthermore, given a multiset $\{t_1, t_2, \ldots, t_p\} \in V(D)$, let $t_i (i = 1, 2, \ldots, p)$ have a how-provenance polynomial $W'_i = W'_{i_1} + W'_{i_2} + \cdots + W'_{i_q}$. If for $\{t_1, t_2, \ldots, t_p\}$ and t, the following condition holds, then one can claim that M is valid for t (not for a single how-provenance monomial): Every monomial W'_{i_j} in $\sum_{i=1}^{p} W'_i$ can be* mapped *to some monomial in W as a one-to-one function under M.*

Example 14. *Continuing Example 10, recall Q_3, V_2, and view mapping $M_{23} = (h_{23}, \phi_{23})$. In terms of* schema-level conditions, *M_{23} is satisfied for all query tuples because 1) V_2 has the* same granularity *as Q_3 under M_{23}; and 2) the aggregate term of V_2, G, can be mapped to the aggregate variable of Q_3, Gid, which also shares the same aggregate function $COUNT$ and thus satisfies Definition 8.*

However, by looking at the provenance-annotated instance of V_2 shown in Table 3.26, the tuple-level condition *does not hold for the query tuple $t_{q_3 2}$. By comparing its provenance (i.e.*

Table 3.26 – $V_2(D)$ with how-provenance

	Ty	COUNT(G)	prov
$t_{v_2 1}$	rRNA	3	$g_2 + g_3 + g_4$

$W = g_2 + g_3$) to the provenance polynomial of the view tuple $t_{v_2 1}$ (i.e. $W' = g_2 + g_3 + g_4$), it is not hard to observe that the monomial mapping between W' and W is not a one-to-one function since g_4 is missing from the mapping. Note that if the predicate in Q_3, $Gid \leq 3$, is relaxed to $Gid \leq 4$ then the token g_4 appears in W and the monomial mapping between W' and W is one-to-one. However, if the predicate is further relaxed to $Gid \leq 5$, then g_5 is included in W and the tuple-level condition is again violated since g_5 is not in W'. This reasoning is significantly more complicated than that in Example 12 since the validity of view mappings is determined by comparing entire how-provenance polynomials between the query tuple and view tuples instead of single how-provenance monomials.

3.2.2.6 Algorithmic details of ProvCite

Based on the validity conditions of the view mappings defined above for PBA, the algorithmic details of ProvCite are presented in this subsection step by step with an overview provided in Algorithm 4.

Algorithm 4: Overview of ProvCite

Input : a set of views: $\mathcal{V} = \{V_1, V_2, ..., V_k\}$, user query: Q, a Database instance D
Output: Valid view mappings $\mathcal{M}(t)$ for every query tuple in $Q(D)$

1 *Preprocessing step*: Return a set of all possible view mappings and the provenance of Q
2 *Reasoning step*: Under a view mapping M from V to Q, determine the validity of M for each query tuple by comparing the provenance between Q and V by following the validity conditions proposed in Section 3.2.2.3 (for conjunctive queries) or Section 3.2.2.4.
3 **return** all valid view mappings $\mathcal{M}(t)$.

Preprocessing step. The major overhead is retrieving the query provenance, which is determined by the underlying provenance-enabled database.

Reasoning step. Let N_{pv} be the total number of how-provenance monomials in the view instance and N_{pq} be the total number of how-provenance monomials in the query instance (this is also the number of tuples before aggregation). Then the time to check the validity of a view mapping is $O(N_{pq} + N_{pv})$ since every how-provenance monomial in the view instance

is compared to some how-provenance monomial in the query instance. If there are m view mappings, then the overall time complexity for this step is $O(m*N_{pq})+O(m*N_{pv})$. Suppose k is an upper bound on the number of relational subgoals in the query or view body, and the largest relation in the database has n tuples. Then the time complexity becomes $O(m*n^k)$. Our experiments with realistic queries, however, show that in practice the performance is still acceptable since both N_{pq} and N_{pv} are typically not very large (less than 1 million).

Example 15. *Given the views V_1-V_8 defined in Example 1, suppose the query is as follows:*

$Q_6(G, COUNT(T), MAX(L), COUNT(E)) :-$
$Exon(E, L, T'), Transcript(T, N, Ty, G), T = T', E \leq 3$

In the pre-processing *step, the provenance of the query is retrieved. Using the instances of* Exon, Gene *and* Transcript *shown in Tables 3.1-3.6, the instance of Q along with the how-provenance polynomials is shown in Table 3.27.*

Table 3.27 – $Q_6(D)$ with how-provenance polynomials

	G	COUNT(T)	MAX(L)	COUNT(E)	prov
$t_{q6}1$	1	1	1	1	$e_1 * r_1$
$t_{q6}2$	2	2	3	2	$e_2 * r_2 + e_3 * r_2$

Table 3.28 – Candidate view mappings for Q_6

view mapping	aggregate terms covered	relational subgoals covered
M_{36}	COUNT(T) (100)	Transcript(T, N, Ty, G) (01)
M_{46}	COUNT(T) (100)	Transcript(T, N, Ty, G) (01)
M_{66}	COUNT(T),MAX(L) , COUNT(E) (111)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)
M_{76}	COUNT(T) (100)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)
M_{86}	MAX(L), COUNT(E) (011)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)

Next, all possible view mappings *are constructed. Five view mappings could be found, i.e., $M_{36} = (h_{36}, \phi_{36})$, $M_{46} = (h_{46}, \phi_{46})$, $M_{66} = (h_{66}, \phi_{66})$, $M_{76} = (h_{76}, \phi_{76})$ and $M_{86} = (h_{86}, \phi_{86})$, under which V_3, V_4, V_6, V_7 and V_8 are mapped to Q_6 respectively. M_{66}, M_{76} and M_{86} have the same form, where h_{66}, h_{76} and h_{86} is $\{Transcript(T1, N1, Ty1, G1) \rightarrow Transcript(T, N, Ty, G), Exon(E, L, T2) \rightarrow Exon(E, L, T')\}$ and ϕ_{66}, ϕ_{76} and ϕ_{86} are the*

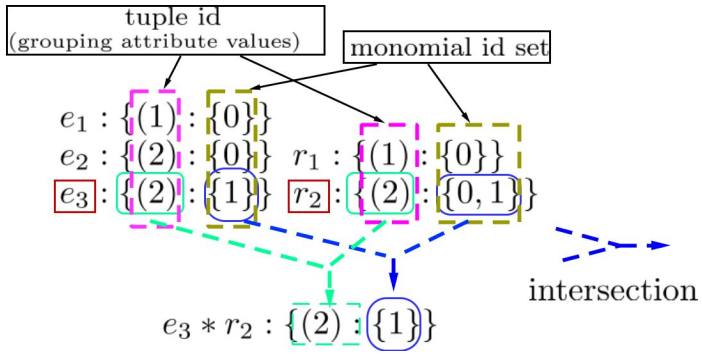


Figure 3.1 – Query provenance index for Q_6 and how to compute coordinate for $e_3 * r_2$ from $V_4(D)$

induced variable mappings. In contrast, M_{36} *and* M_{46} *only map the subgoal* $Transcript(T, N, Ty, G)$ *from* V_3 *and* V_4 *to* $Transcript(T, N, Ty, G)$ *from* Q_6 *respectively. Note that all the* schema-level conditions *are independent of the individual query tuples, and can be used to remove invalid view mappings early. In this example, all the five view mappings satisfy the* schema-level conditions*, since under each mapping all the grouping variables and at least one aggregate term of* Q_6 *are* covered*. Table 3.28 shows how each view mapping covers the aggregate terms and relational subgoals of* Q_6 *(ignore the bit arrays for now).*

In the Reasoning *step, since* V_4 *and* V_6 *are conjunctive views, the validity of* M_{46} *and* M_{66} *for a query tuple* t_q *only depends on the* existence *of* t_q *in* $V_4(D)$ *and* $V_6(D)$ *under mapping* M_{46} *and* M_{66}*. So one can simply retrieve the base relation tuple for each provenance token appearing in the query to evaluate the view predicates. For example, the validity of* M_{66} *can be checked simply by examining the predicates of* V_6*. Since the first predicate in* V_6*,* $T = T'$*, is also in* Q_6*, every tuple in the query instance must satisfy it. However, the second predicate,* $E \leq 2$*, can affect the* existence *of query tuples in* V_6*. Table 3.20 shows that only the tuples with how-provenance tokens* e_1 *and* e_2 *satisfy* $E \leq 2$*. Thus* M_{66} *is only valid for* t_{q61}*, whose how-provenance polynomial only includes* e_1*. Note that the implementation used here is different from RBA since evaluating view predicates is achieved by referencing the base relation tuples with the query provenance rather than computing extra predicates in the query evaluation.*

Table 3.29 – $V_3(D)$ with how-provenance polynomials

	G	COUNT(T)	prov
$t_{v_3 1}$	1	1	r_1
$t_{v_3 2}$	2	1	r_2

Table 3.30 – $V_7(D)$ with how-provenance polynomials

	G1	COUNT(T1)	prov
$t_{v_7 1}$	1	1	$e_1 * r_1$
$t_{v_7 2}$	2	1	$e_3 * r_2 + e_4 * r_2$

Table 3.31 – $V_8(D)$ with how-provenance polynomials

	G1	MAX(L)	COUNT(E)	prov
$t_{v_8 1}$	1	1	1	$e_1 * r_1$
$t_{v_8 2}$	2	3	2	$e_2 * r_2 + e_3 * r_2 + e_4 * r_2$

Table 3.32 – $Q_6(D)$ with valid view mappings

	G	COUNT(T)	MAX(L)	COUNT(E)	valid view mappings
$t_{q_6 1}$	1	1	1	1	$M_{36}, M_{66}, M_{76}, M_{86}$
$t_{q_6 2}$	2	2	3	2	M_{36}

In contrast, since V_3, V_7 and V_8 are aggregate views, it is thus essential to compare their how-provenance expressions with the how-provenance of the query to check the tuple-level conditions. That can be naively implemented by scanning the entire query provenance for every view mapping *to build satisfiable provenance mappings (Def. 10), which is expensive when the query provenance and view provenance are large. To reduce this cost, the following optimization strategies are proposed to speed up the process of producing valid view mappings.*

3.2.2.7 Optimizations in the implementations

Query provenance index optimization. To build mappings between the provenance tokens of the query tuples and the one of the view tuples, one naive solution is to go through *all* query provenance monomials attached to the query instances for a provenance monomial from the view instances to check whether the one-to-one monomial mappings defined in Definition 10 can be constructed or not, which, however, is undeniably expensive especially for large query and view instances. To reason about valid view mappings in an efficient manner by using PBA such that multiple scans over query provenance can be avoided, an

index I could be built for each token in the query provenance to indicate which query tuples (represented by grouping variable values) and which provenance monomials the token is in, which is exemplified as below:

Example 16. *Continuing Example 15, the provenance index for Q_6 is shown in Figure 3.1. For example, referencing Table 3.27, note that token r_2 is in the 0^{th} and 1^{st} monomial in the query tuple $t_{q_6}2$, which has value 2 for the grouping variable G. So the index for r_2 is $r_2 : \{(2) : \{0, 1\}\}$, where (2) represents the tuple id while $\{0, 1\}$ is the monomial id set. For a how-provenance monomial in the view, e.g. $e_3 * r_2$ in $t_{v_7}2$ (with grouping variable value 2), to determine whether it can be mapped to some query provenance monomial, one can retrieve the index for e_3 and r_2 with grouping variable value 2 respectively, i.e. $\{1\}$ and $\{0, 1\}$, and take the intersection, i.e. $\{1\}$. This indicates that e_3 and r_2 coexist in the 1^{st} monomial of the query tuple with grouping variable value 2 (i.e. t_{q2}). This derivation process is highlighted in Figure 3.1.*

In the end, valid view mappings for every query tuple are presented in Table 3.32. Note that for tuple $t_{q_6}2$, although all of its how-provenance monomials exist in the view tuple $t_{v_8}2$, it does not include $e_4 * r_2$ which is used to construct $t_{v_8}2$, violating the tuple-level condition. Intuitively, since the value of the aggregate term may come from this component of the monomial ($e_4 * r_2$), $t_{v_8}2$ should not provide citation information for $t_{q_6}2$.

The intersection operation can be further optimized by representing the monomial ids with bit arrays, where the i^{th} bit is 0/1 iff a token is/isn't in the i^{th} monomial, and applying bit AND operations. This strategy only requires one full scan over the query provenance to build an index for all view mappings. Details on how to use the index to determine whether provenance mappings from view tuples to query tuple satisfy Def. 10 are presented in Algorithm 5.

Materialization and parallelism optimization. To further improve performance, the provenance of the aggregate views along with the view content can be materialized before the query arrives. The strategy with materialized view provenance is called *eager*, whereas that without is called *lazy*. The *eager* and *lazy* strategies are compared experimentally.

Algorithm 5: Checking provenance mappings

Input : a view V, a query Q, a view mapping M from V to Q, query tuple $t_q \in Q(D)$, query provenance index I, a set of view tuples $T_v \subseteq V(D)$
Output: Whether provenance mappings from T_v to t_q satisfy Def. 10

```
for t_v ∈ T_v do
    Retrieve the provenance monomial set P̄ of t_v Retrieve the grouping variable values (Gv) of Q
     under mapping M
    for each provenance monomial P ∈ P̄ do
        for each provenance token p ∈ P do
            if p is not in I OR Gv is not in the entry of I for p then
                return false
            end
        end
        Perform intersection over the index for p with grouping variable values Gv
        if the intersection result is empty then
            return false
        end
    end
end
return true
```

It is not hard to observe that reasoning about the validity of view mappings is highly parallelizable since the reasoning between different view mappings is independent. However, since fully parallel computation in a single machine can incur large memory consumption, ProvCite only processes five view mappings at a time. Exploring how to fully develop our system in a distributed environment is left for future work.

3.3 Citation generation

3.3.1 Covering sets

Given a set of valid view mappings $\mathcal{M}(t)$ for a query tuple t derived by either RBA or PBA, the next step is to combine those view mappings to construct *covering sets*, which provide instructions on how to combine the citation information associated with the corresponding views to construct citations. The formal definition of *covering sets* is presented as below.

Definition 11. Covering set *Let $C \subseteq \mathcal{M}(t)$ be a set of valid view mappings. Then C is a covering set of view mappings for t from the instance of a query Q iff*

1. *if Q is a conjunctive query, then:*

- *No $V \in \mathcal{M}(t) \setminus C$ can be added to C to cover more subgoals of Q or variables in $\bar{X}$; and*
- *No $V \in C$ can be removed from C and cover the same subgoals of Q and variables in $\bar{X}$.*

2. *if Q is an aggregate query, then:*
   - *No $V \in \mathcal{M}(t) \setminus C$ can be added to C to cover more subgoals of Q or* aggregate terms *in $\bar{X}$; and*
   - *No $V \in C$ can be removed from C and cover the same subgoals of Q and* aggregate terms *in $\bar{X}$.*

Note that for each tuple t there may be a *set* of covering sets, $\{C_1, ..., C_k\}$.

Example 17. *Returning to Example 16, based on the definition of covering sets provided above and covered aggregate terms by each individual view mappings shown in Table 3.28, the covering sets for $t_{q_6}1$ are $C_1 = \{M_{66}\}$, $C_2 = \{M_{76}, M_{86}\}$ and $C_3 = \{M_{36}, M_{86}\}$. Note that other combinations of view mappings such as $\{M_{36}, M_{76}, M_{86}\}$ are duplicates since removing either M_{36} or M_{76} results in the view mapping combinations collectively mapping the same set of aggregate terms of Q_6 as $\{M_{36}, M_{76}, M_{86}\}$.*

Table 3.33 – Instance of Q_6 with covering sets

	G	valid view mappings	covering sets
$t_{q_6}1$	1	$M_{36}, M_{66}, M_{76}, M_{86}$	$\{\{M_{66}\}, \{M_{76}, M_{86}\}, \{M_{86}, M_{36}\}\}$
$t_{q_6}2$	2	M_{36}	$\{\{M_{36}\}\}$

For the resulting covering set $C_i = \{M_1, M_2, \ldots, M_l\}$, the citation views corresponding to each view mapping are *jointly* used (denoted *) to construct a citation for t, denoted $M_1 * M_2 * ... * M_l$. The citations from each C_i are then *alternately* used (denoted $+^R$) to construct a citation for t, denoted as $C_1 +^R \cdots +^R C_p$.

3.3.2 Optimizations to computing covering sets

Bit array optimization. The computation of covering sets involves merging valid view mappings and removing duplicates, which can be optimized using bit operations. For example, for Q_6 in Example 15-17, the aggregate term $COUNT(T)$, $MAX(L)$ and $COUNT(E)$ are covered by $\{M_{36}, M_{66}, M_{76}\}$ (denoted by S_1), $\{M_{66}, M_{86}\}$ (denoted by S_2) and $\{M_{66}, M_{86}\}$ (denoted by S_3) respectively (see Table 3.28). One can encode the $0^{th} - 3^{rd}$ view mappings M_{36}, M_{66}, M_{76} and M_{86} as $\{0, 1, 2, 3\}$, the $0^{th} - 2^{nd}$ aggregate terms ($COUNT(T)$, $MAX(L)$ and $COUNT(E)$) as $\{0, 1, 2\}$, and the $0^{th} - 1^{st}$ relational subgoals ($Exon$ and $Transcript$) as $\{0, 1\}$. In this manner, arbitrary view mapping combinations (and thus covering sets) can be represented using three bit arrays in which the i^{th} bit is 1 (0) iff the i^{th} view mapping is included (missing), or the i^{th} aggregated term or relational subgoal is covered (not covered). For example, M_{76} is the 2^{nd} view mapping, represented by 0010 (the leftmost bit is the 0^{th} bit). M_{76} covers the 0^{th} aggregate term ($COUNT(T)$) and the 0^{th} and 1^{st} relational subgoals, which are represented by bit arrays 100 and 11 respectively. The bit array representations for other view mappings are listed in Table 3.28. To compute covering sets, the view mapping combinations from the cross product of $\{S_1, S_2, S_3\}$ (denoted by $S_1 \times S_2 \times S_3$) are considered, which are constructed by applying bit OR operations over the bit arrays from those view mappings. For example, referencing Table 3.28, the covering set $\{M_{76}, M_{86}\}$ can be constructed by unioning bit arrays 0010 and 0001, and the aggregate terms (relational subgoals resp.) jointly covered by them are computed by unioning 100 and 011 (11 and 11 resp.). The pseudocode for computing covering sets using bit array representations is presented in Algorithm 6.

Clustering algorithm optimization. Since cross product ($\times$) is commutative and associative, different orderings of operands result in the same output but may incur different overhead. For example, with $S_1 \times S_2 \times S_3$, if $S_2 \times S_3$ is computed first, the result is $\{M_{66}, M_{86}\}$, $\{M_{86}, M_{86}\}$, $\{M_{66}, M_{66}\}$, $\{M_{86}, M_{66}\}$. After removing obvious redundancy, the result is $\{M_{66}, M_{86}\}$, $\{M_{86}\}$, $\{M_{66}\}$. Note that $\{M_{66}, M_{86}\}$ is a duplicate compared to $\{M_{66}\}$ since 1) $\{M_{66}, M_{86}\}$ and $\{M_{66}\}$ cover the same set of aggregate terms and relational

Table 3.34 – Binary encoding for the view mappings of Q_6

view mapping	aggregate terms covered	relational subgoals covered
M_{36} (1000)	COUNT(T) (100)	Transcript(T, N, Ty, G) (01)
M_{66} (0100)	COUNT(T),MAX(L) , COUNT(E) (111)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)
M_{76} (0010)	COUNT(T) (100)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)
M_{86} (0001)	MAX(L), COUNT(E) (011)	Transcript(T, N, Ty, G) ,Exon(E, L, T') (11)

Algorithm 6: Compute covering sets

Input : a set of valid view mappings $\mathcal{M}$ for query tuple $t \in Q(D)$, query Q
Output: a set of covering sets C

```
1  For each aggregate term of Q, derive a set of view mappings covering it, which forms an array of
     view mapping sets S.
2  Determine the order to compute the cross product of every element in S
3  Initialize C as the first view mapping set s_0 from S.
4  for each set s ∈ S − {s_0} do
5      Initialize cross product result C' = {}:
6      for each view mapping set M̄' ∈ C do
7          for each view mapping M ∈ s do
8              get three bit arrays of M̄' (M): b_1 (b'_1), b_2(b'_2) and b_3 (b'_3)
9              construct new view mapping set M̄'' based on the bit OR operation result
                 b_i ∨ b'_i(i = 1, 2, 3) and put it into C'
10         end
11     end
12     Remove duplicates from C'
13     C = C'
14 end
```

subgoals (checked by comparing the corresponding bit arrays); and 2) $\{M_{66}\}$ is a subset of $\{M_{66}, M_{86}\}$. It is therefore safe to remove $\{M_{66}, M_{86}\}$ since in the final result, any view mapping combinations which include $\{M_{66}, M_{86}\}$ will be a duplicate compared to one that includes $\{M_{66}\}$ and thus won't be a covering set. So the intermediate result of $S_2 \times S_3$ is $\{\{M_{66}\}, \{M_{86}\}\}$, which is smaller than the result of the other pairs. This is due to the high similarity between S_2 and S_3 (actually $S_2 = S_3$). To find good orderings for computing the cross product such that the intermediate result is minimized, clustering algorithms are applied so that view mapping sets which are similar to each other can be clustered and merged first (e.g. S_2 and S_3). In ProCite, the affinity propagation clustering algorithm [121] is used since it does not require a pre-specified number of clusters.

3.3.3 Policy to generate citations

To construct citations based on the derived covering sets, it is essential to utilize the citation information extracted from the views whose view mappings appear in those covering sets. But prior to that, the interpretations on the operators $+^R$ and $*$ are essential, which is crucial to organize the citation information from citation views in a reasonable manner to meet the DBAs' needs.

There are two possible interpretations on the $*$ operator, i.e. *join* and *union*. The former one conducts SQL-like join operations, i.e. combining all the elements with the same keys together from two JSON-formatted citations while the latter one simply unions two JSON-formatted citations together.

There are two possible ways to evaluate $+^R$, i.e., *union* and *min*. The *union* of covering sets is straightforward, although it can lead to very large citations. In contrast, the goal of *min* is to find the covering set with minimum cost (according to some custom cost function), and it is evaluated as the covering sets are being constructed. It therefore has the advantage of avoiding enumerating all covering sets, and thus reducing the overhead of this step in all three approaches. Note that the problem of finding a min-cost covering set can be formalized as a set cover problem, which is NP-complete. However, a greedy algorithm can be applied to derive an $O(log\ n)$-approximate solution [122], which is presented in Algorithm 7.

Algorithm 7: Greedy algorithm using cost function

Input : A set of valid view mapping $\mathcal{M}(t)$, a set of maximally covered relation $MCR(t)$.the view set $\mathcal{V}$
Output: A set of view mappings $\mathcal{C}(t)$
`/* create a set to contain the selected view mappings, the result of which should be the covering set with minimal cost */`
Initialize $C(t) = \{\}$
Derive a set of maximally covered distinguished variables $MCH(t)$
while $MCR(t) \neq \Phi$ *and* $MCH(t) \neq \Phi$ **do**
  `/* use` $Hv(M)$ `and` $R(M)$ `to denote the distinguished variables and relations that view mapping` M `covers in the query */`
  select M from $\mathcal{M}(t)$ that can minimize $\frac{cost(M)}{|Hv(M) \bigcap MCH(t)| + |R(M) \bigcap MCR(t)|}$
  $MCH(t) = MCH(t) \backslash Hv(M)$, $MCR(t) = MCR(t) \backslash R(M)$
  add M to $C(t)$
end
return $C(t)$

Intuitively, the cost function is designed such that the covering set with the smallest number of view mappings in the $*$-term has the lowest cost, balanced by the number of *unmatched* terms, including unmatched subgoals, unmatched distinguished variables, and unmatched lambda terms in each view. For example, for the query tuple t_{q61} (see Table 3.33), the covering set $\{M_{66}\}$ is only composed of one view mapping while all the other covering sets include more than one view mappings, resulting in smaller cost of the covering set $\{M_{66}\}$ than others. However, if one lambda term $\lambda G1$ is added in V_6, there is no matched conditions such as $G = 1$ in the query (recall that under the view mapping $M_{66} = (h_{66}, \phi_{66})$, $\phi_{66}(G1) = G$) which can explicitly evaluate the lambda variable in V_6, thus incurring more overhead to evaluate the lambda terms in V_6 and therefore adding more cost to the covering set $\{M_{66}\}$. In the case where $+^R$ is evaluated as *union*, it is named as *full case*. Otherwise, it is named as *min case*. The trade-offs between the two different choices of interpretations are also experimentally studied.

3.4 Experimental evaluations

3.4.1 Experiment setups

TLA, SSLA and ProvCite are implemented in Java 8 and used PostgreSQL 9.6.3 as the underlying DBMS. All experiments were conducted on a linux server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz and 64GB of central memory. Our code is publicly available [1] and uses pieces of code developed by the authors of [123]. Note that the query provenance and view provenance are essential for reasoning in ProvCite. Although there are provenance tools which support aggregate queries for relational database systems, e.g. GProM [124], they are overly complex for our purposes, and become a bottleneck for interactive computation. I therefore implemented a provenance layer from scratch, which simply collects how-provenance [7, 8] for each query tuple.

[1] Our code is available at `https://github.com/thuwuyinjun/Data_citation_demo`.

Datasets Our experiments used four datasets, i.e. the GtoPdb database [2], DBLP-NSF dataset[3][125], GENCODE dataset [63] and Hetionet dataset[4].

The GtoPdb database is a searchable database with information on drug targets and the prescription medicines and experimental drugs that act on them. The database content is organized by a hierarchy of families of drug targets. The total number of targets is 2825. Each family of targets is curated by a (potentially different) group of experts. Information about a family is presented to users via a web-page view of the database, associated with a family-specific citation. The citation is generated from hard-coded SQL queries in the web-page form that retrieve the appropriate snippets of information from the database, such as contributors and/or curators, which are then formatted to create a citation.

DBLP-NSF is developed by us which connects computer science publications—extracted from DBLP—to their NSF funding grants—extracted from the National Science Foundation grant dataset. The idea was to add funding information to traditional paper citations, and to be able to vary between citing the conference proceedings (if large number of papers from the same conference were in the result set) and citing individual papers. DBLP-NSF consists of 17 relations (authors, papers, grants, etc.). Author is the largest relation with about six million tuples, and the average size across all relations is about 0.6 million tuples.

GENCODE dataset is an encyclopedia of genes and gene variants whose goal is to identify all functional elements in the human genome using annotations. The gene annotation process involves a combination of automatic annotation, manual annotation, and experimental validation. For genes that are manually annotated, information is maintained about the responsible research groups. Statistics are also provided for every gene – an *aggregate view* over the genes – which has another type of citation giving credit to the creators of the aggregate view. Common queries over GENCODE also involve aggregation. For instance, one query computes statistics for every *type* of gene. After loading this dataset into relational DBMS, there are 7 relations with 600K tuples in each table on average.

[2]GtoPdb is available at `http://www.guidetopharmacology.org/download.jsp`.
[3]`https://data.mendeley.com/datasets/ycnngyv5bd`
[4]`https://neo4j.het.io/browser/`

Hetionet is a database that "encodes" biology by integrating various types of biological information from different publicly available resources [119, 64]. As data is copied from these source datasets, citation information (generally in the form of traditional publication IDs) is also copied and should be propagated to the results of queries. The majority of queries against this database involve aggregation to retrieve statistical information. The original version of Heitonet is stored in Neo4j, which is converted into relational database for the experiments, ending up with 38 relations with 38K tuples in each relation on average.

Workloads There are two types of workloads used in the experiments, i.e. *synthetic workloads* and *realistic workloads*. For synthetic workloads, conjunctive queries were built using a *user query generator* which takes as input: 1) the number of relational subgoals; 2) the number of tuples in the extended query result (N_t). I also implemented a *view generator* which takes as input: 1) the number of views (N_v); 2) the number of lambda terms in total (N_l); and 3) the total number of predicates (N_p) to generate random conjunctive views. Both query generators and view generators are extended for aggregate queries and aggregate views by taking into considerations the total number of how-provenance monomials in the query instance (N_{pq}) and in the view instance (N_{pv}). Each generated view has a single citation query attached to it. In the experiments, the configurations of the query generator and view generator ensure that there is only one view mapping from each view to the query.

For the *realistic workloads*, I use frequent realistic queries against the four databases, and build views to represent the portions of data in the database associated with predefined citations, which are illustrated as below:

For GtoPdb database, I used citation views and general *conjunctive queries* ($Q0 - Q7$) from anticipated workloads of GtoPdbḞor GtoPdb, general queries were designed by consulting with the database owners, and views were designed based on its web-page views. For each view, the corresponding citation query is the query used to generate the hard-coded citations on the web-page.

For DBLP-NSF, six conjunctive views are created, each of which is associated with 1-2 citation queries to correspond to citations to a single paper, single conference and single

grant. General *conjunctive user queries* ($Q8-Q10$) simulate cases where users are interested in papers from certain authors, certain conferences and certain years together with the grant information of those papers. I also add aggregate views to reflect publicly available statistics related to this database, such as the total number of publications per faculty member[5] and the total number of grants per institution[6]. Some *realistic aggregate queries* are designed to represent other summary information, such as the total number of publications per institution ($q1$) and total amount of grants per state ($q2$).

To represent the summary information provided by GENCODE, I defined aggregate views to compute the total number of transcripts per gene, and the total number of exons per gene and per transcript. Two additional parameterized views are also defined to represent basic information (e.g. ID, name and type) for each transcript and gene, respectively. The realistic *aggregate queries* compute the total number of exons ($q3$) and the total number of transcripts per type of gene ($q4$) respectively.

Hetionet integrates information from various resources, and includes information about genes, biological process, drugs, etc. This information is stored in different relations in the database. Of these, the biological process relation is associated with citation information (i.e. related publication IDs). After consulting with the authors of Hetionet, two views were defined. The first one is a parameterized view showing the biological processes that a particular gene is involved in. The second counts the total number of connections between each biological process and corresponding genes by joining several relations, such as the biological process and gene relations. A typical *aggregate query* ($q5$) counts the total number of connections between each biological process and a certain drug via some genes.

3.4.2 Experimental results

In the experiments, I primarily record the total execution time for constructing covering sets, i.e. t_{total} by using TLA, SSLA and ProvCite respectively, in which the major overhead includes the time to query the instance or the provenance from the underlying database

[5] `http://csrankings.org/`

[6] `https://dellweb.bfa.nsf.gov/awdlst2/default.asp`

instances, t_q and the time to reason about covering sets, t_{cs}. For ProvCite, I also explore the effect of the query provenance index and materialization of the view provenance (proposed in Section 3.2.2.7) on the time performance. The effect of the optimization strategies for deriving covering sets (proposed in Section 3.3.2) is also studied in this subsection.

3.4.2.1 Synthetic experimental results on conjunctive queries

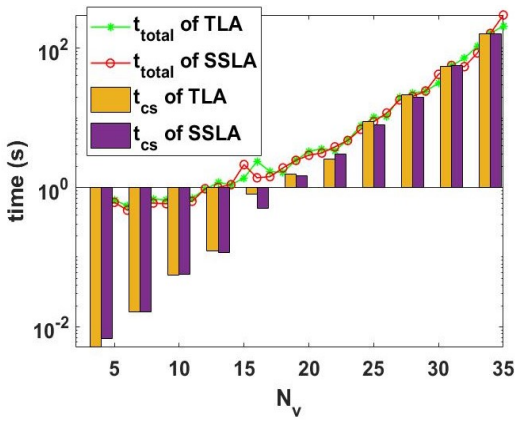
Exp1 The first experiment evaluates the effect of N_v on time performance and citation size. I configured the query generator to generate queries which produce a result set of about one million tuples ($N_t = 10^6$) using a subset of the product of four randomly selected relations. The view generator varied the number of views and ensured valid view mappings from each view to the query. Here, I do not consider other view features such as lambda terms and predicates.

Results. For the *full* case, thousands of covering sets are generated as the number of view mappings exceeds 30. As expected, the corresponding time to generate them (t_{cs}) increases exponentially. As shown in Figure 3.2a, the major overhead in t_{total} is the reasoning time t_{cs} when N_v exceeds 25, leading to a convergence of the three approaches. However, it is worth mentioning that according to the realistic scenarios, the number of matched view mappings won't exceed 20; thus, t_{total} in the three approaches will be less than 30 seconds (Figure 3.2a), which is reasonable response time.

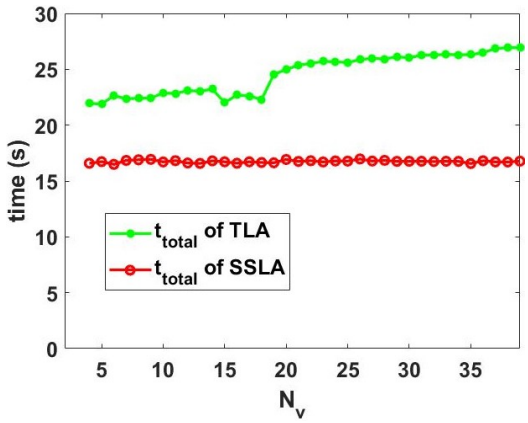
Figure 3.2b shows the results for the *min* case, which has a huge speed-up compared to the *full* case. Notice that even with a large N_v, t_{total} is acceptable (about 25 seconds).

These results reveal the effect of N_v on the time performance. In the *full* case, exponentially large covering sets are generated, taking up to 10 minutes as N_v becomes large. Since each covering set represents a possible citation, this also generates thousands of citations. On the other hand, the *min* case returns the "best" citation, which reduces t_{cs} to a few milliseconds and leads to a steady t_{total} as N_v increases.

Exp2. This experiment tests how N_p influences the performance and size of citations. Like Exp1, the query generator randomly picked four relations and ensured $N_t = 10^6$. However,



(a) t_{total} and t_{cs} in *full* case (log scale in Y-axis)



(b) t_{total} in *min* case

Figure 3.2 – time performance VS number of view mappings

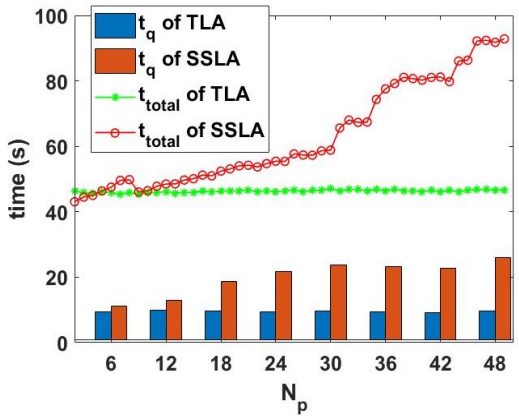


Figure 3.3 – t_{total} and t_q VS N_p in *min* case

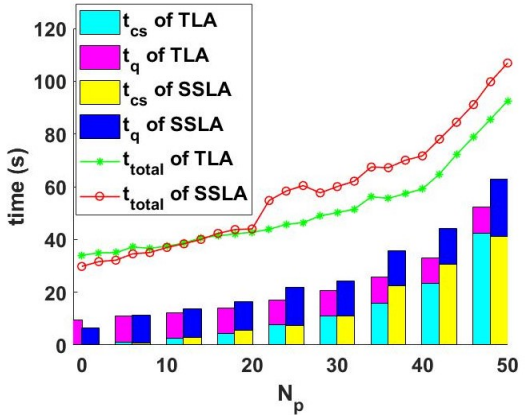


Figure 3.4 – t_{total}, t_q and t_{cs} VS N_p in *full* case

the view generator fixed the number of views as 15 and varied the total number of local predicates (and thus N_p) from 0 to 50.

Results. The number of predicates (N_p) influences the time performance of TLA and SSLA in two ways. First, more predicates add more complexity to the extended query and thus increases the time to execute the query (t_q). Second, it can create more groups in the extended query result, incurring more reasoning time (t_{cs}).

Figures 3.3 and 3.4 show how t_{total} and its major timing components (t_q and t_{cs}) are influenced by N_p in the *min* and *full* cases, respectively. In Figure 3.3, t_q is included for

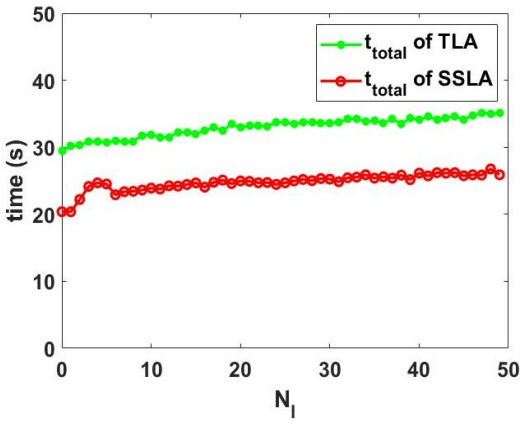


Figure 3.5 – t_{total} VS N_l in *full* case

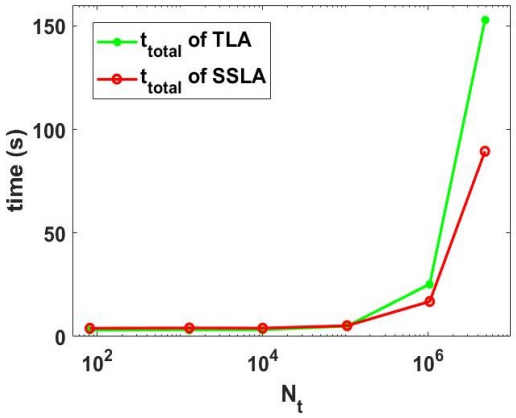


Figure 3.6 – t_{total} VS N_t in *full* case (log scale in X-axis)

Table 3.35 – Experimental results on real workloads (full case)

Query	N_t	N_v	N_p	N_{cs}	t_{total} in TLA (s)	t_{total} in SSLA (s)
Q0	8868	1	0	1	0.25	0.18
Q1	1366	1	0	1	0.19	0.15
Q2	2522	7	6	1	0.25	0.21
Q3	120	8	6	1	0.18	0.16
Q4	5748	7	6	7	0.26	0.22
Q5	1	8	6	1	0.16	0.14
Q6	271	7	6	1	0.17	0.16
Q7	521	1	0	1	0.16	0.14
Q8	4884	4	1	3	1.19	1.18
Q9	27	4	1	3	1.81	1.72
Q10	7	2	0	2	0.94	0.91

TLA and SSLA, and shows that in the *min* case, increasing N_p results in slight increases in the (extended) query execution time for SSLA; thus t_{total} in SSLA is only about twice that in TLA when N_p is up to 50. The slightly worse performance of SSLA is due to the complexity of the extended query. Recall that the boolean values of local predicates are explicitly evaluated and then used for grouping in SSLA, which is not necessary in TLA. The same is true for the *full* case (Figure 3.4).

For the *full* case, the reasoning time t_{cs} becomes a major overhead as N_p increases for both TLA and SSLA. This is because more predicates can create more groups in the query result, which incurs more t_{cs} in total.

As a result, this can explain the effect of N_p on the time performance.

Exp3 Theoretically, in all three approaches, when there are more lambda terms, more attributes must be returned in the extended query in order to evaluate citations for the parameterized views. This should increase the query execution time t_q. However, Figure 3.5 shows that the number of lambda terms N_l has almost no effect on the overall performance of the derivation process t_{total}, of which t_q is a component. The *full* case shows similar results (ignored here). This can addresses our concern on the effect of lambda terms defined in the citation views.

Exp4. This experiment evaluates the scalability of our approaches in terms of time performance by varying N_t. The view generator randomly generates 15 views ($N_v = 15$) with randomly assigned local predicates and lambda terms. The query generator randomly generates a query with four relations but varies the result size (from 10^2 to 10^7) at each iteration.

Results. Figure 3.6 shows that when there are fewer than 10^7 tuples in the query result, the time to calculate covering sets (t_{total}) in TLA and SSLA is less than 200 seconds, which can thus explain the influence of N_t on the performance.

3.4.2.2 Realistic experimental results on conjunctive queries

I now report experiments performed on the realistic datasets. Table 3.35 shows all experimental results for the *full* case. Results for the *min* case are only marginally better, and are not shown.

Exp5 This experiment evaluates how well the proposed approaches handle realistic workloads in the GtoPdb dataset. In this experiment, 14 views were created and each view has one associated citation query according to the web-page views. Eight user queries (Q0-Q7) were collected from the owners of GtoPdb.

Results. The first eight rows of Table 3.35 shows the result of Exp5. The time to generate covering sets t_{cs} for all the queries is very small (less than 1 second). Although there are 14 views in total, only one covering set exists for most queries and the number of view mappings is far fewer than 14, leading to the short response time. This is the case

when the views partition the relations.

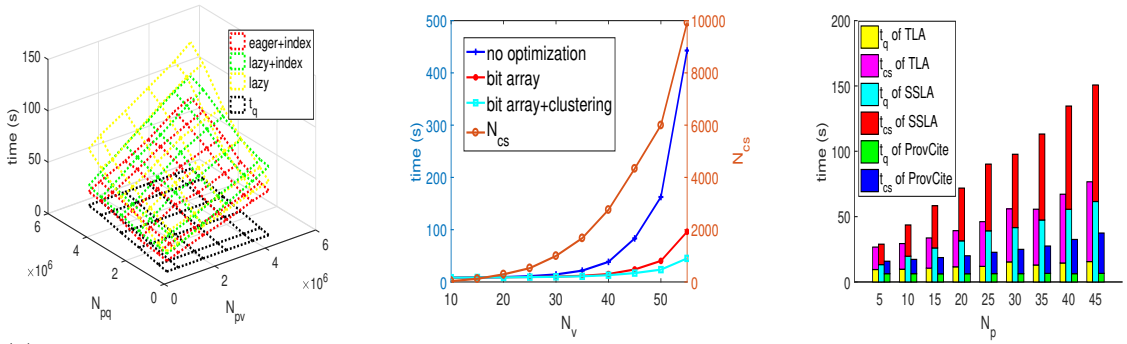
Exp6 This experiment is conducted by executing conjunctive queries ($Q8 - Q10$) on the DBLP-NSF dataset. $Q8$ asks for the titles of papers in a certain conference (e.g. VLDB), while $Q9$ retrieves the titles of all papers published by a given author in a given year. These correspond to searches over the DBLP dataset where users are interested in papers of specific authors or conferences. $Q10$ returns the NSF grants that support papers in a given conference (e.g. VLDB).

Results. The last three rows of Table 3.35 show that the number of covering sets N_{cs} and the time to generate them t_{total} are still very small. Thus Exp5 and Exp6 provide answers to the time performance in the realistic scenarios.

3.4.2.3 Synthetic experimental results on aggregate queries

Exp7. This experiment measures how the total execution time (t_{total}) is influenced by the total number of how-provenance monomials in the query instance (N_{pq}) as well as in the view instance (N_{pv}). I randomly generate an aggregate query, and vary N_{pq} by adding appropriate predicates. A fixed number of aggregate views are also generated such that there is exact one view mapping from each of them to the query and the total number of view mappings is fixed at 20. In practice, the number of views that touch the query is usually far smaller than the total number of views, so 20 is a pretty large number. Both N_{pq} and N_{pv} are varied from 50K to 5M. The total time is measured for different (N_{pq}, N_{pv}) pairs under the *eager* and *lazy* strategy with the query provenance index, and the *lazy* strategy without the index.

Results. The results are shown using 3D surfaces in Figure 3.7a, with the eager strategy with index, lazy strategy with index and pure lazy strategy shown in red, green and yellow respectively. The query time t_q is also recorded in black. It shows that the query provenance index leads to about 1.0x-1.8x speed-ups in most cases by comparing lazy strategy with index and pure lazy strategy, while materializing view provenance results in about 1.1x-1.5x speed-ups by comparing eager strategy and lazy strategy with index. The combination of



(a) t_{total} with varied N_{pq} and N_{pv}

(b) t_{total} with varied N_v

(c) t_{total} with varied N_p

Figure 3.7 – Experimental results for synthetic workloads

the index and eager strategy leads to up to 2x performance gains. The result also shows the *scalability* of our approach since it takes less than 2 mins to process a query instance with up to 5 million how-provenance monomials and 20 views with up to 5 million how-provenance monomials, which rarely happens in practice.

Exp8. The goal of this experiment is to compare the relative performance of ProvCite, TLA and SSLA while varying the number of view mappings (N_v). Since TLA and SSLA cannot handle aggregate views, only conjunctive views are used. In this case, the provenance of views is not necessary; there is no difference between the eager and lazy strategy and the query provenance index is not useful. However, the two optimization strategies on covering set computation, i.e. applying bit arrays and clustering algorithms, are useful and are measured here. The query is a fixed aggregate query with 1 million how-provenance monomials in its instance. N_v is varied from 1 to 50 and there are no predicates or lambda variables for each individual view.

Results. The experimental results are presented in Figure 3.7b, which shows the change of t_{total} for ProvCide and the number of covering sets (N_{cs}) as the number of view mappings (N_v) increases, with and without using bit arrays and clustering algorithm. TLA and SSLA have almost the same performance as ProvCite, and are not shown. Figure 3.7b shows that when N_v is large, an exponential number of covering sets are generated, leading to bad performance (see blue line). Bit array computations and the use of clustering leads to about

Table 3.36 – Summary of datasets

Dataset name	relation #	average tuple # per relation	tuple # of largest relation
GENECODE	7	600k	2000k
Hetionet	38	60k	500k
DBLP-NSF	17	600k	6000k

Table 3.37 – Experimental results on realistic datasets

Query	t_{total} (s) (eager + index)	t_{total} (s) (lazy + index)	t_{total} (s) (lazy)	N_{pq}	N_v	N_p	t_q(s)
$q1$	4.95	6.62	6.44	507k	2	0	2.92
$q2$	5.90	6.49	6.33	416k	1	0	2.80
$q3$	11.05	12.93	11.89	1237k	1	0	5.09
$q4$	1.75	2.06	3.26	203k	2	0	0.69
$q5$	4.65	5.10	4.81	243k	3	0	2.32

a 5x and 2x speed-up respectively; an order of magnitude performance gain is achieved by combining both.

Exp9. In this experiment, ProvCite is compared with TLA and SSLA while varying the total number of predicates (N_p) in views. Similar to Exp2, the query is an aggregate query which can generate about 1 million tuples. The number of view mappings is fixed at 10 and there are initially no predicates. In each run, one more local predicate is added. As shown in [12], increasing N_p significantly influences the query time and hence performance of TLA and SSLA since the query is extended to evaluate the view predicates.

Results. The results are shown in Figure 3.7c. As the number of predicates increases, $t_{total} = t_{cs} + t_q$ increases slowly for ProvCite. In contrast, TLA and SSLA are twice as slow as ProvCite for large N_p. To understand the reason for this, the query time for TLA, SSLA and ProvCite is also presented in this figure, showing that the increasing query time becomes the major overhead for both TLA and SSLA.

3.4.2.4 Realistic experimental results on aggregate queries

The experimental results for realistic workloads are presented in Table 3.37, which includes the total execution time (t_{total}) for three cases (lazy, lazy + index and eager + index), as well as the metrics that can potentially affect the performance: the total number of how-provenance monomials in the query instance (N_{pq}), the total number of view mappings (N_v),

the total number of predicates in the views under all the view mappings (N_p) and the time to query the provenance along with the instance. Except for $q3$, t_{total} is less than 10 seconds for all queries. Although N_{pq} is more than one million in $q3$, t_{total} is only about 11-13 seconds for the three strategies, which is acceptable considering the large query instance. Note that the index does not always help since it may take significant time to build the index for query provenance (e.g. up to 3 seconds for $q3$) and its performance gain is not significant in the case of small number of view mappings.

However, as shown in Section 3.4.2.3, the index provides scalability especially in the extreme cases. I also list the query time over the provenance-enabled database in the last column, which indicates that the reasoning time ($t_{cs} = t_{total} - t_q$) is almost the same as t_q. Thus, while users are browsing the query result, the system can generate covering sets for all query tuples in the background, and instantly construct formatted citations upon tuple selection.

3.5 Acknowledgement

For this part of the dissertation, I want to acknowledge the many collaborators who helped from all over the world: Dr. Peter Buneman initially framed the data citation problem, and proposed many new related ideas. Drs. Val Tannen, Daniel Deutch and Tova Milo helped form the connection between data citation and data provenance. Dr. Abdussalam Alawini helped architect CiteDB, the implementation of our data citation system, along with Dr. Gianmaria Silvello, who also created connections to related work in the Digital Libraries community.

CHAPTER 4: Incrementally updating machine learning models using provenance

As mentioned in Chapter 1, incrementally updating machine learning models after deleting the training samples of interest is a crucial step for many data science applications, e.g., evaluating the importance of training samples for a certain model. One straightforward method is to always retrain the model from scratch to reflect the changes of the training set, which, however, is computationally expensive especially when continuous deletion requests are required in some online applications. As a consequence, it is technically challenging on how to *efficiently* update the model parameters in light of the deletions of one or multiple training samples. However, the state-of-the-art works either target simple ML models, or require explicit changes over the existing training algorithms, thus not appropriate for ML models constructed by the widely used gradient descent method (GD) and its variants, i.e. stochastic gradient descent method (SGD) and mini-batch gradient descent method (mb-SGD for short).

So in this set of my work, I target designing algorithms to update *general* ML models constructed by *SGD/GD method* after the deletions of small subsets of the training samples. After some initial studies, I proposed two different types of solutions, which are discussed in this section. Before the detailed discussions, I provided notations used in the following discussion first.

4.1 Preliminary

The training set $\{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^n$ has n samples and each sample has m_0 features, which means that the matrix $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$ is an $n \times m$ matrix. The loss or objective function

for a general machine learning model is defined as:

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} F(\mathbf{w}, \mathbf{z}_i)$$

where $\mathbf{w}$ represents a vector of the model parameters and $F(\mathbf{w}, \mathbf{z}_i)$ is the loss for the i-th sample. The gradient and Hessian matrix of $F(\mathbf{w})$ are

$$\nabla F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla F(\mathbf{w}, \mathbf{z}_i), \quad \mathbf{H}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{H}(\mathbf{w}, \mathbf{z}_i)$$

Suppose the model parameter is updated through mini-batch stochastic gradient descent (SGD):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{B_t}\sum_{i\in\mathcal{B}_t} \nabla F(\mathbf{w}_t, \mathbf{z}_i) = \mathbf{w}_t - \eta_t \nabla F(\mathbf{w}_t, \mathcal{B}_t) \tag{4.1}$$

where $\mathcal{B}_t$ is a randomly sampled mini-batch of size B, η_t is the *learning rate* at the t^{th} iteration and $\nabla F(\mathbf{w}_t, \mathcal{B}_t)$ averages the gradients evaluated on all the training samples in $\mathcal{B}_t$. For GD, $\mathcal{B}_t$ includes all the training samples ($B = n$). As the special case of SGD, the update rule of gradient descent (GD) is $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t/n \sum_{i=1}^{n} \nabla F(\mathbf{w}_t, \mathbf{z}_i)$.

For example, the objective functions of linear regression, binary logistic regression and multinomial logistic regression with $L2-$regularization are presented in Equations 4.2-4.4 respectively [1]

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\mathbf{w})^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{4.2}$$

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\ln(1 + \exp\{-y_i\mathbf{w}^\top\mathbf{x}_i\}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{4.3}$$

[1] Here I assume that the two possible labels in binary logistic regression are 1 and -1.

$$F(\mathbf{w}) = \frac{1}{n}\sum_{k=1}^{q}\sum_{y_i=k}(\ln(\sum_{j=1}^{q}e^{\mathbf{w}_j^\top \mathbf{x}_i}) - \mathbf{w}_k^T\mathbf{x}_i) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$
$$\mathbf{w} = vec([\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q]) \tag{4.4}$$

where $\mathbf{w}$ is the vector of model parameters and λ is the *regularization rate*. For simplicity, I denote $\mathbf{w} = vec([\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q])$ for multinomial logistic regression where q represents the number of possible classes.

By absorbing the regularization terms into the summation terms, the objective functions of linear regression, binary logistic regression and multinomial logistic regression over the i_{th} sample $(i = 1, 2, \ldots, n)$ are presented in Equations 4.5-4.7 respectively:

$$F(\mathbf{w}, \mathbf{z}_i) = (y_i - \mathbf{x}_i^T\mathbf{w})^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{4.5}$$

$$F(\mathbf{w}, \mathbf{z}_i) = \ln(1 + \exp\{-y_i\mathbf{w}^\top\mathbf{x}_i\}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{4.6}$$

$$F(\mathbf{w}, \mathbf{z}_i) = \sum_{k=1}^{q}\mathbb{1}(y_i = k)\cdot(\ln(\sum_{j=1}^{q}e^{\mathbf{w}_j^\top \mathbf{x}_i}) - \mathbf{w}_k^T\mathbf{x}_i) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$
$$\mathbf{w} = vec([\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_q]) \tag{4.7}$$

Considering the similarities between binary logistic regression and multinomial logistic regression and the complexity of the computation related to the latter one, I will only present the formulas related to binary logistic regression below. All the theorems that hold for binary logistic regression can be also proven to be true for multinomial logistic regression.

Then the update rules of linear regression and binary logistic regression by using the

mini-batch SGD become:

$$\mathbf{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\mathbf{w}_t - \frac{2\eta_t}{|\mathcal{B}_t|}\sum_{i\in\mathcal{B}_t}\mathbf{x}_i(\mathbf{x}_i^T\mathbf{w}_t - y_i) \tag{4.8}$$

$$\mathbf{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\mathbf{w}_t + \frac{\eta_t}{|\mathcal{B}_t|}\sum_{i\in\mathcal{B}_t} y_i\mathbf{x}_i(1 - \frac{1}{1+\exp\{-y_i\mathbf{w}_t^T\mathbf{x}_i\}}) \tag{4.9}$$

4.2 Provenance-based ML model updates

Note that the incremental ML model update problem is quite similar to the classical materialized view maintenance problem in the database community where provenance semiring model [10, 24] is applied to incrementally propagate the effect of removing base relation tuples that some views rely on to the corresponding view instances. So one natural idea is to develop solutions for propagating the deletions of training samples to ML models by using the extended provenance-semiring model from [25] over the linear algebraic operators in the update rule of the SGD/GD method. Therefore, in this section, I will introduce my solution PrIU and its optimized version PrIU-opt to incrementally update regression models, e.g. linear regression model, logistic regression model and possibly other *generative additive models* [28], which explicitly utilize the provenance-semiring model extended for linear algebra operations. In what follows, this section starts by the detailed descriptions of this extended provenance-semiring model mentioned above [25].

4.2.1 Provenance semiring model for linear algebra operators [25]

To provide provenance support for linear algebra operations, this extended provenance-semiring model starts by annotating each row and each column in the input matrix with unique provenance tokens, where the input matrix represents the input feature matrix of the training dataset to some training algorithm. Since only the deletions of training samples are considered in the ML model update problem, only a simpler version of this provenance semiring model is considered, where only each row of the input matrix, i.e. each sample $\mathbf{x}_i$, is annotated with a unique provenance token p_i. Therefore, each provenance-annotated sample

becomes a *tensor product*, $p_i * \mathbf{x}_i$, where $*$ is used to concatenate the provenance expression and the matrix expression. Based on this annotations, the resulting tensor product after the matrix addition and multiplication operations between two provenance-annotated input matrices, $r_1 * \mathbf{M}_1$ and $r_2 * \mathbf{M}_2$, are:

$$\texttt{Matrix Addition} : r_1 * \mathbf{M}_1 + r_2 * \mathbf{M}_2 \tag{4.10}$$

$$\texttt{Matrix Multiplication} : (r_1 * \mathbf{M}_1) \cdot (r_2 * \mathbf{M}_2) = (r_1 r_2) * (\mathbf{M}_1 \mathbf{M}_2) \tag{4.11}$$

in which the operator $+$ and $\cdot$ defined over tensor products are associative and commutative and I assume that the dimensions of $\mathbf{M}_1$ and $\mathbf{M}_2$ are compatible with each other in the matrix addition or multiplication operations. In addition, the two operators are distributive, i.e.:

$$(r_1 * \mathbf{M}_1 + r_2 * \mathbf{M}_2) \cdot (r_3 * \mathbf{M}_3 + r_4 * \mathbf{M}_4) \tag{4.12}$$

$$= (r_1 r_3) * \mathbf{M}_1 \mathbf{M}_3 + (r_1 r_4) * \mathbf{M}_1 \mathbf{M}_4 + (r_2 r_3) * \mathbf{M}_2 \mathbf{M}_3 + (r_2 r_4) * \mathbf{M}_2 \mathbf{M}_4 \tag{4.13}$$

Then all tensor products $p_i * \mathbf{x}_i$ from the input training data are propagated through matrix additions and multiplications to the final output by using the computational rules in Equation (4.11) - (4.12), which ends up with a tensor product, representing the provenance expression of the output result.

Example 18. *Suppose there are 4 training samples $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ and $\boldsymbol{x}_4$, which are annotated by provenance token p_1, p_2, p_3, p_4 respectively, given the following linear algebra program:*

$$\boldsymbol{w} = 4\boldsymbol{x}_1^T \boldsymbol{x}_2 \boldsymbol{x}_3 + 3\boldsymbol{x}_1^T \boldsymbol{x}_2 \boldsymbol{x}_4 + 5\boldsymbol{x}_1^T \boldsymbol{x}_3 \boldsymbol{x}_4 + \boldsymbol{x}_2^T \boldsymbol{x}_3 \boldsymbol{x}_4$$

The resulting tensor product is:

$$\mathscr{W} = (4p_1p_2p_3) * \boldsymbol{x}_1^T \boldsymbol{x}_2 \boldsymbol{x}_3 + (3p_1p_2p_4) * \boldsymbol{x}_1^T \boldsymbol{x}_2 \boldsymbol{x}_4 + (5p_1p_3p_4) * \boldsymbol{x}_1^T \boldsymbol{x}_3 \boldsymbol{x}_4 + (p_2p_3p_4) * \boldsymbol{x}_2^T \boldsymbol{x}_3 \boldsymbol{x}_4 \tag{4.14}$$

With the tensor product expression for the resulting matrix, to delete the effect of some training sample $\mathbf{x}_i$, I set the corresponding token p_i as 0_{prov}, representing the absence of the $\mathbf{x}_i$, while all the other tokens are set as 1_{prov}, representing the presence of the corresponding samples, in which 0_{prov} and 1_{prov} have the following properties:

$$0_{\text{prov}} * \mathbf{M} = \mathbf{0} \tag{4.15}$$

$$1_{\text{prov}} * \mathbf{M} = \mathbf{M} \tag{4.16}$$

$$p0_{\text{prov}} = 0_{\text{prov}} \tag{4.17}$$

$$p1_{\text{prov}} = p \tag{4.18}$$

in which $\mathbf{M}$ and p represent an arbitrary matrix and an arbitrary provenance token respectively. Equation (4.15) (and Equation (4.16) resp.) tells us that any tensor product $0_{\text{prov}} * \mathbf{M}$ ($1_{\text{prov}} * \mathbf{M}$ resp.) results in zero matrix (and the matrix $\mathbf{M}$ itself resp.). In contrast, Equation (4.17) and Equation (4.18) are about the effect of 0_{prov} and 1_{prov} within every provenance monomial, which indicate that 0_{prov} zeros out the entire provenance monomial while 1_{prov} does not have any effect on other portions of the provenance monomial.

Example 19. *Let us revisit Example 18. If users want to remove the sample $\boldsymbol{x}_2$ and retain all the other samples in the final output, then p_2 is set as 0_{prov} while all the other provenance tokens are set as 1_{prov}. Therefore, the tensor product $\mathscr{W}$ in Equation (4.14) becomes:*

$$\begin{aligned}\mathscr{W} = (1_{\text{prov}}0_{\text{prov}}1_{\text{prov}}) * (4\boldsymbol{x}_1^T\boldsymbol{x}_2\boldsymbol{x}_3) + (1_{\text{prov}}0_{\text{prov}}1_{\text{prov}}) * (3\boldsymbol{x}_1^T\boldsymbol{x}_2\boldsymbol{x}_4) \\ + (1_{\text{prov}}1_{\text{prov}}1_{\text{prov}}) * (5\boldsymbol{x}_1^T\boldsymbol{x}_3\boldsymbol{x}_4) + (0_{\text{prov}}1_{\text{prov}}1_{\text{prov}}) * \boldsymbol{x}_2^T\boldsymbol{x}_3\boldsymbol{x}_4\end{aligned} \tag{4.19}$$

Then by using the computation rule in Equation (4.15)-(4.18), the formula above becomes:

$$\mathscr{W} = 5\boldsymbol{x}_1^T\boldsymbol{x}_3\boldsymbol{x}_4 \tag{4.20}$$

which is actually the same as the result of $\boldsymbol{w}$ after $\boldsymbol{x}_2$ is removed.

4.2.2 Constructing tensor products for SGD/GD update rules

In this subsection, I introduce how provenance is annotated in the update rules of linear regression and logistic regression respectively.

4.2.2.1 Constructing tensor products for SGD/GD update rules of liner regression model

As Section 4.2.1 mentioned, each training sample $\mathbf{x}_i$ can be annotated with a unique provenance token p_i such that the tensor product for the resulting model parameters can be derived by applying the computation rule in Equation (4.11)-(4.18) to the matrix addition and multiplication operators in the SGD/GD update rules and iteratively propagate the resulting tensor product expression until model parameters convergence, which can successfully transform the SGD/GD update rules of linear regression, i.e. Equation (4.8), to its provenance-annotated version, i.e.:

$$\mathcal{W}_{t+1} \leftarrow [(1-\eta_t\lambda)(1_k * \mathbf{I}) - \frac{2\eta_t}{\mathcal{P}_t}\sum_{i\in\mathcal{B}_t} p_i^2 * \mathbf{x}_i\mathbf{x}_i^T]\mathcal{W}_t + \frac{2\eta_t}{\mathcal{P}_t}\sum_{i\in\mathcal{B}_t} p_i^2 * \mathbf{x}_i y_i \tag{4.21}$$

where $\mathcal{W}_t$ represents the provenance-annotated expression for the vector $\mathbf{w}_t$ of model parameters while $\mathcal{P}_t$ represents a provenance-annotated expression for the number of samples in the min-batch $\mathcal{B}_t$, for example, following the approach to aggregation in [8], $\mathcal{P}_t = \sum_{i\in\mathcal{B}_t} p_i * 1$.

In the semiring framework there is no division operation so I only focus on the provenance annotated formula, $\sum_{i\in\mathcal{B}_t} p_i^2 * \mathbf{x}_i\mathbf{x}_i^T$ and attempt to leverage the provenance semiring model described in the last subsection for incrementally updating this formula. Specifically, in this formula, by setting the provenance tokens of all the remaining samples as 1_{prov} and the provenance tokens of all the removed samples as 0_{prov}, the SGDGD update rule becomes:

$$\mathbf{w}_{t+1}^U \leftarrow [(1-\eta_t\lambda)(\mathbf{I}) - \frac{2\eta_t}{B_t^U}\sum_{i\in\mathcal{B}_t, i\notin\mathcal{R}} \mathbf{x}_i\mathbf{x}_i^T]\mathbf{w}_t^U + \frac{2\eta_t}{B_t^U}\sum_{i\in\mathcal{B}_t, i\notin\mathcal{R}} \mathbf{x}_i y_i \tag{4.22}$$

which is thus the update rule of linear regression model parameters after the deletions of samples from $\mathcal{R}$.

However, there is still one way to work around the division operations in the provenance annotated formula in Equation (4.21). To accomplish this, I can replace $\mathcal{P}_t$ with an integer to represent the number of remaining samples at current mini-batch after the removal of some samples, in which case, the provenance tokens for the remaining training samples and removed training samples are set as 1_{prov} and 0_{prov} respectively and the number of the updated mini-batch size is set as B_t^U. By denoting the set of the removed sample as R and applying the computation rule in Equation (4.11)-(4.18), the provenance-annotated update rule for $\mathcal{W}_{t+1}$ becomes:

$$\mathcal{W}_{t+1}^U \leftarrow [(1-\eta_t\lambda)(1_{\text{prov}} * \mathbf{I}) - \frac{2\eta_t}{B_t^U}\sum_{\substack{i\in\mathcal{B}_t \\ ,i\notin\mathcal{R}}} p_i^2 * \mathbf{x}_i\mathbf{x}_i^T]\mathcal{W}_t^U + \frac{2\eta_t}{B_t^U}\sum_{\substack{i\in\mathcal{B}_t \\ ,i\notin\mathcal{R}}} p_i^2 * \mathbf{x}_i y_i \tag{4.23}$$

4.2.2.2 Constructing tensor products for SGD/GD update rules of logistic regression model

Unfortunately, there exists one obvious obstacle toward annotating provenance to the GD or SGD update rules of logistic regression model since the provenance model from [25] only handles the linear algebra operators, ignoring the ubiquitous use of the non-linear operations in ML training algorithms. For example, the function $f(x) = 1 - \frac{1}{1+\exp^{-x}}$ is used in the SGD/GD update rule of logistic regression, i.e. Equation (4.9), which is a non-linear operation, taking the product $y_i\mathbf{w}_t^T\mathbf{x}_i$ as the input argument x. Therefore, as the first step to apply the provenance model in Equation (4.9), I determined to linearize the non-linear operations in Equation (4.9) by utilizing *piecewise linear interpolation* (see [26] for 1D piecewise linear interpolation and see [126] for multi-dimension piecewise linear interpolation, which is useful for interpolating the update rule of multi-nomial logistic regression), which has the following properties:

Lemma 1. ***Piecewise linear interpolation*** *In Piecewise linear interpolation [26], I as-*

sume that the function to be approximated is a continuous function $f(x)$ *where* $x \in [a, b]$. *Piecewise linear interpolation starts by picking up a series of* breaking points, x_i *such that* $a < x_1 < x_2 < \cdots < x_p < b$ *and then constructs a linear interpolant* $s(x)$ *over each interval* $[x_{j-1}, x_j)$ *as follows:*

$$\begin{aligned} s(x) &= \frac{x - x_{j-1}}{x_j - x_{j-1}} f(x_j) + \frac{x_j - x}{x_j - x_{j-1}} f(x_{j-1}) \\ &= a_j x + b_j, x \in [x_{j-1}, x_j) \end{aligned} \tag{4.24}$$

The following property holds on how close the value of $s(x)$ *is compared to the original function* $f(x)$*:*

$$\begin{aligned} |f(x) - s(x)| &\leq \frac{1}{8}(\Delta x)^2 \max_{a \leq x \leq b} |f''(x)| = O((\Delta x)^2) \\ |f'(x) - s'(x)| &\leq \frac{1}{2}(\Delta x) \max_{a \leq x \leq b} |f''(x)| = O((\Delta x)) \end{aligned} \tag{4.25}$$

in which $\Delta x = \max\{x_i - x_{i-1}\}_{i=1}^n$. □

Throughout the paper, I will consider the case in which the variable x in $f(x)$ is defined within an interval $[-a, a]$ $(a = 20)$ that is equally partitioned into 10^6 sub-intervals; for x outside $[-a, a]$, I assume that $s(x)$ is a constant since when $|x| > a$, the value of $f(x)$ is very close to its bound (0 or 1). I will show that the length of each sub-interval influences the approximation rate. As a consequence, after the interpolation step over Equation (4.9), the approximated update rule becomes:

$$\mathbf{w}_{t+1}^L \approx [(1 - \eta_t \lambda)\mathbf{I} + \frac{\eta_t}{B} \sum_{i \in \mathcal{B}_t} a_{i,t} \mathbf{x}_i \mathbf{x}_i^T] \mathbf{w}_t^L + \frac{\eta_t}{B} \sum_{i \in \mathcal{B}_t} b_{i,t} y_i \mathbf{x}_i \tag{4.26}$$

in which $\mathbf{w}_t^L$ represents the model parameter after linearization at t_{th} iteration and $a_{i,t}, b_{i,t}$ are the linear coefficients produced by the linearizations, which depends on which sub-interval (defined by piecewise linear interpolation) the value of $y_i \mathbf{w}_t^T \mathbf{x}_i$ locates and thus should be varied between different $\mathbf{x}_i$ and different $\mathbf{w}_t$ (see the associated subscript).

Then after the linearization step, by taking similar steps to the update rule of linear

regression after the deletions of training samples, $\mathcal{R}$, the provenance-annotated version of Equation (4.26) is shown as below:

$$\mathcal{W}_{t+1}^{LU} \leftarrow [(1-\eta_t\lambda)(1_{\text{prov}} * \mathbf{I}) + \frac{\eta_t}{B_t^U} \sum_{\substack{i \in \mathcal{B}_t \\ ,i \notin \mathcal{R}}} p_i^2 * (a_{i,t}\mathbf{x}_i\mathbf{x}_i^T)]\mathcal{W}_t^{LU} + \frac{\eta_t}{B_t^U} \sum_{\substack{i \in \mathcal{B}_t \\ ,i \notin \mathcal{R}}} p_i^2 * (b_{i,t}y_i\mathbf{x}_i) \tag{4.27}$$

Then by setting all the p_i in Equation (4.27) as 1_{prov}, I can get the update rule for the updated model parameter $\mathbf{w}_t^{LU}$, i.e.:

$$\mathbf{w}_{t+1}^{LU} \approx [(1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{B_t^U} \sum_{\substack{i \in \mathcal{B}_t \\ ,i \notin \mathcal{R}}} a_{i,t}\mathbf{x}_i\mathbf{x}_i^T]\mathbf{w}_t^{LU} + \frac{\eta_t}{B_t^U} \sum_{\substack{i \in \mathcal{B}_t \\ ,i \notin \mathcal{R}}} b_{i,t}y_i\mathbf{x}_i \tag{4.28}$$

4.2.3 Theoretical analysis

Although the update rule on the tensor products for the model parameters has been provided in the previous subsection, there are still some theoretical questions unresolved, including whether the tensor product expressions will be converged eventually and whether the linearized update rule of logistic regression is a good approximation of the original update rule, which will be answered in this subsection.

In terms of the first concern, i.e. the convergence concern, it is closely related to the classical convergence problems for SGD/GD method, which have been extensively studied in the machine learning community [127, 128, 129, 130, 131]. In [131], convergence conditions have been provided for GD and SGD over strongly convex objective functions, which are presented as below:

Lemma 2. ***Convergence conditions for general mb-SGD.*** *[131] Given an objective function $f(\boldsymbol{w})$, which is $L-$Lipschitz continuous and $\lambda-$strongly convex once the learning rate η_t satisfies: 1) $\eta_t < \frac{1}{L}$; 2) η_t is a constant across all the iterations (denoted by η), then $\boldsymbol{w}_t$ converges when mb-SGD is used.*

Note that the convergence conditions above can exactly fit linear regression and logistic regression with L2-regularization because their objective functions are strongly convex. Then

in terms of the convergence issue for the provenance expression $\mathcal{W}_t^U$ in Equation (4.23) and $\mathcal{W}_t^{LU}$ in Equation (4.27), ideally, I expect that they can be converged when the original model parameter $\mathbf{w}_t$ converges. To prove or disprove this argument, a clear definition of the convergence of the tensor products is required first, which is provided as below.

Definition 12. Convergence of provenance-annotated expressions. *The expression* $\mathcal{W}_t = \sum_i p_{i,t} * \boldsymbol{u}_{i,t}$ *converges when* $t \to \infty$ *iff every matrix* $\boldsymbol{u}_{i,t}$ *converges when* $t \to \infty$.

Unfortunately, my theoretical analysis shows that there is no convergence guarantee for $\mathcal{W}_t^U$ and $\mathcal{W}_t^{LU}$ under the convergence conditions from Lemma 2, i.e.:

Theorem 1. $\mathcal{W}_t^U$ *in Equation* (4.23) *and* $\mathcal{W}_t^{LU}$ *in Equation* (4.27) *need not converge under the conditions in Lemma 2.*

However, $\mathcal{W}_t^U$ in Equation (4.23) and $\mathcal{W}_t^{LU}$ in Equation (4.27) converge under the conditions in Lemma 2 with one more assumption about the provenance expression, i.e.:

Theorem 2. *The expectation of* $\mathcal{W}_t^U$ *in Equation* (4.23) *and of* $\mathcal{W}_t^{LU}$ *in Equation* (4.27)*, converge when* $t \to \infty$ *if I also assume that provenance polynomial multiplication is* idempotent.

Intuitively speaking, the assumption of multiplication idempotence for provenance polynomials means that I do not track *multiple joint uses of the same data sample*, which is not problematic for deletion propagation.

Then in terms of the concerns on the closeness between the results of the linearized update rule (i.e. $\mathbf{w}_t^L$ in Equation (4.26)) and the one of the original update rule of logistic regression (i.e. $\mathbf{w}_t$ in Equation (4.9)), I provide rigorous theoretical analysis on the distance between $\mathbf{w}_t$ and $\mathbf{w}_t^L$ as below by following the approximation property of piecewise linear interpolation shown in Lemma 1, i.e.:

Theorem 3. $||E(\boldsymbol{w}_t - \boldsymbol{w}_t^L)||_2$ *is bounded by* $O((\Delta x)^2)$ *where* Δx *is an arbitrarily small value representing the length of the longest sub-interval used in piecewise linear interpolations.*

Furthermore, in terms of the updated model parameters for logistic regression, it is also essential to guarantee that the updated parameters $\mathbf{w}_t^{LU}$ are close to the real updated model parameters without linearization (denoted by $\mathbf{w}^{RU}$), i.e.:

$$\mathbf{w}_{t+1}^{RU} \leftarrow (1-\eta_t\lambda)\mathbf{w}_t^{RU} + \frac{\eta_t}{B_t^U}\sum_{\substack{i\in\mathcal{B}_t \\ ,i\notin\mathcal{R}}} y_i\mathbf{x}_i f(y_i\mathbf{w}_t^{RU}\mathbf{x}_i) \tag{4.29}$$

Recall that $f(x) = 1 - \frac{1}{1+e^{-x}}$. Note that the linear coefficients $a_{i,t}$ and $b_{i,t}$ in Equation (4.28) are actually derived in the training phase where all samples exist (rather than in the model update phase), which implies that a larger difference between $\mathbf{w}_t^{LU}$ and $\mathbf{w}_t^{RU}$ should be expected. Surprisingly, I can prove that the distance between $\mathbf{w}_t^{LU}$ and $\mathbf{w}_t^{RU}$ is still small enough.

Theorem 4. *$||E(\boldsymbol{w}_t^{LU} - \boldsymbol{w}_t^{RU})||_2$ is bounded by $O(\frac{\Delta n}{n}\Delta x) + O((\frac{\Delta n}{n})^2) + O((\Delta x)^2)$, where Δn is the number of the removed samples and Δx is defined in Theorem 3.*

4.2.4 PrIU for linear regression

	BaseL	PrIU	PrIU-opt
linear regression	$O(T(B-\Delta B)m)$	$O(T(zm+\Delta Bm))$	$O(\min\{\Delta n, m\}m^2) + O(Tm)$
logistic regression	$O(T((B-\Delta B)m + C_{non}m))$	$O(Tzm) + O(T\Delta Bm)$	$O(t_s(zm+\Delta Bm)) + O(\min\{\Delta n, m\}m^2) + O((T-t_s)m)$

Table 4.1 – Summary of the time complexity of BaseL, PrIU and PrIU-opt

This subsection will be centered around the implementation details of PrIU for linear regression and logistic regression based on the provenance-annotated update rules for model

	PrIU	PrIU-opt
linear regression	$O(Tzm)$	$O(m^2)$
logistic regression	$O(Tzm) + O(n\lceil\frac{TB}{n}\rceil)$	$O(m^2) + O(t_szm) + O(n\lceil\frac{t_sB}{n}\rceil)$

Table 4.2 – Summary of the space complexity of PrIU and PrIU-opt for caching provenance information

parameters introduced in Section 4.2.2. As mentioned before, there is a great concern on the non-negligible overhead of maintaining and using provenance in database domain, which becomes more severe for provenance for ML training algorithms due to its iterative computation.

To alleviate this issue, I utilize a series of optimization strategies from linear algebra domain to minimize the overhead of using provenance and speed up the incremental updates on ML models. The time complexity and space complexity of the proposed incremental update solutions and the baseline approach, i.e. the approach to retraining from scratch, (denoted as BaseL) are also provided for comparison. First of all, as mentioned in Section 4.2.2, by setting all the remaining provenance tokens as 1_{prov} after minor deletions on the training datasets, Equation (4.23) becomes Equation (4.22), which can be further rewritten as follows:

$$\mathbf{w}_{t+1}^U \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{B_t^U}\sum_{i\in\mathcal{B}_t}\mathbf{x}_i\mathbf{x}_i^T + \sum_{\substack{i\in\mathcal{B}_t \\ ,i\in\mathcal{R}}}\mathbf{x}_i\mathbf{x}_i^T]\mathbf{w}_t^U + \frac{2\eta_t}{B_t^U}(\sum_{i\in\mathcal{B}_t}\mathbf{x}_i y_i - \sum_{i\in\mathcal{B}_t,i\in\mathcal{R}}\mathbf{x}_i y_i) \quad (4.30)$$

which simply separates the contributions of the full minibatch from the deleted samples. In this formula, the term $\sum_{i\in\mathcal{B}_t}\mathbf{x}_i\mathbf{x}_i^T$ and $\sum_{i\in\mathcal{B}_t}\mathbf{x}_i y_i$ are not affected by the model parameters and used for the training process on the full training dataset, thus cached during the training phase before the deletions occur. Then during the incremental update phase, only the term $\sum_{i\in\mathcal{B}_t,i\in\mathcal{R}}\mathbf{x}_i\mathbf{x}_i^T$ needs to be calculated from scratch, which will be very efficient once the number of removed samples within a minibatch is far smaller than the minibatch size. I can further rewrite Equation (4.30) in matrix form, i.e.:

$$\mathbf{w}_{t+1}^U \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{B_t^U}\mathbf{X}_{\mathcal{B}_t}^T\mathbf{X}_{\mathcal{B}_t} + \Delta\mathbf{X}_{\mathcal{B}_t}^T\Delta\mathbf{X}_{\mathcal{B}_t}]\mathbf{w}_t^U + \frac{2\eta_t}{B_t^U}(\sum_{i\in\mathcal{B}_t}\mathbf{x}_i y_i - \sum_{i\in\mathcal{B}_t,i\in\mathcal{R}}\mathbf{x}_i y_i) \quad (4.31)$$

where I use $\mathbf{X}_{\mathcal{B}_t}$ and $\Delta\mathbf{X}_{\mathcal{B}_t}$ to represent the matrix consisting of training samples within the minibatch $\mathcal{B}_t$ and the removed training samples in $\mathcal{B}_t$ respectively. Then let us analyze the complexity of the incremental update strategy above. By utilizing the associativity prop-

erty of matrix multiplication, expensive matrix-matrix multiplications in Equation (4.31) (i.e. $\Delta\mathbf{X}_{\mathcal{B}_t}^T\Delta\mathbf{X}_{\mathcal{B}_t}$) can be avoided, which can be replaced by more efficient matrix-vector multiplications (e.g. computing $\Delta\mathbf{X}_{\mathcal{B}_t}\mathbf{w}_{t+1}^U$ first and then multiplying the result by $\Delta\mathbf{X}_{\mathcal{B}_t}^T$). Then based on this idea, the time complexity to compute $\mathbf{w}_t^U$ by using Equation (4.31) and leveraging the cached terms $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T$ and $\sum_{i\in\mathcal{B}_t}\mathbf{x}_i y_i$ at each iteration will be $O(\Delta Bm+m^2)$ if the number of removed sample is supposed to be ΔB on average across all iterations (recall that the number of feature is m_0 for each sample). In contrast, the time complexity to compute $\mathbf{w}_t^U$ by retraining from scratch (i.e. using Equation (4.22)) without utilizing any cached terms will be $O((B-\Delta B)m)$.

However, note that caching the term $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T$ can incur very high overhead if the feature number, m_0, is very large. Plus, when m_0 is large enough, especially larger than B, Bm will be smaller than m^2, indicating that using Equation (4.22) will incur less time overhead than using Equation (4.31), thus leading to inefficient incremental updates. To address this issue, one can observe that the dimension of the term $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T$ can be reduced by using SVD, i.e. $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T = \mathbf{U}_t\mathbf{S}_t\mathbf{V}_t^T$, where $\mathbf{S}_t$ is a diagonal matrix whose diagonal elements represent the singular values, while $\mathbf{U}_t$ and $\mathbf{V}_t$ are the left and right singular vectors. By retaining only the first z largest singular values and the corresponding singular vectors, $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T$ can be approximated by $\mathbf{U}_{t,1..z}\mathbf{S}_{t,1..z}\mathbf{V}_{t,1..z}^T$ ($\mathbf{U}_{t,1..z}, \mathbf{V}_{t,1..z}$ represents the submatrix composed of the first z columns and $\mathbf{S}_{t,1..z}$ is a diagonal matrix composed of the first z eigenvalues in $\mathbf{S}_t$). Therefore, Equation (4.31) is further rewritten as:

$$\begin{aligned}\mathbf{w}_{t+1}^U \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{B_t^U}(\mathbf{U}_{t,1..z}\mathbf{S}_{t,1..z}\mathbf{V}_{t,1..z}^T \\ - \Delta\mathbf{X}_{\mathcal{B}_t}^T\Delta\mathbf{X}_{\mathcal{B}_t})]\mathbf{w}_t^U + \frac{2\eta_t}{B_t^U}(\sum_{i\in\mathcal{B}_t}\mathbf{x}_i y_i - \sum_{\substack{i\in\mathcal{B}_t \\ ,i\in\mathcal{R}}}\mathbf{x}_i y_i)\end{aligned} \tag{4.32}$$

which incurs overhead $O(zm+\Delta Bm)$ for each iteration. Since $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T = \sum_{i\in\mathcal{B}_t}\mathbf{x}_i\mathbf{x}_i^T$, which sums up B outer products between rank-one vectors, thus having rank $B \leq m$ when $B < m$. So the number of the main component r should be smaller than B. Hence using Equation (4.32) for incremental updates is more efficient than the method to retrain from

scratch which has overhead $O((B-\Delta B)m)$. One requirement to use Equation (4.32) is that the matrix $\mathbf{U}_{t,1..z}, \mathbf{S}_{t,1..z}$ and $\mathbf{V}_{t,1..z}$ should be cached, which incurs $O(zm)$ space for each iteration. If there are T iterations in total, then the total time complexity and space complexity by using Equation (4.32) are $O(T(zm+\Delta Bm))$ and $O(Tzm)$ respectively while the time complexity of retraining from scratch is $O(T(B-\Delta B)m)$.

Note that since $\mathbf{X}_{\mathcal{B}_t}\mathbf{X}_{\mathcal{B}_t}^T$ is approximated by its dimension-reduced version, $\mathbf{U}_{t,1..z}\mathbf{S}_{t,1..z}\mathbf{V}_{t,1..z}^T$, the approximation rate is also analyzed and presented as below:

Theorem 5. ***Approximation ratio*** *Under the convergence conditions for* $\boldsymbol{w}^{(t)}$, $||\boldsymbol{w}^{(t)}||$ *should be bounded by some constant* C. *Suppose* $\frac{||\boldsymbol{U}_{t,1..z}\boldsymbol{S}_{t,1..z}\boldsymbol{V}_{t,1..z}^T||_2}{||\boldsymbol{U}_t\boldsymbol{S}_t\boldsymbol{V}_t^T||_2} \geq 1-\epsilon$ *where* ϵ *is a small value, then the change of model parameters caused by the approximation will be bounded by* $O(\epsilon)$.

This shows that with proper choice of r in the SVD approximation, the updated model parameters computed by PrIU or PrIU-opt should be still very close to the expected result. So in my implementations, z is chosen based on ϵ (say 0.01) such that the inequality in Theorem 5 is satisfied.

Optimizations I provided an optimized version of PrIU, PrIU-opt. when the feature number m_0 is small, for which I utilize the GD-version update rules, i.e.:

$$\mathbf{w}_{t+1} \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T]\mathbf{w}_t + \frac{2\eta_t}{n}\sum_{i=1}^{n}\mathbf{x}_i y_i \tag{4.33}$$

$$\mathbf{w}_{t+1}^U \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{n-\Delta n}\sum_{i\notin\mathcal{R}}\mathbf{x}_i\mathbf{x}_i^T]\mathbf{w}_t^U + \frac{2\eta_t}{n-\Delta n_t}\sum_{i\notin\mathcal{R}}\mathbf{x}_i y_i \tag{4.34}$$

in which Δn represents the size of the removed sample set $\mathcal{R}$ and recall that $\mathbf{w}$ and $\mathbf{w}^U$ represent the model parameters before and after the deletions of the training sample set $\mathcal{R}$ respectively. By representing the training sample matrix $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$ as $\mathbf{X}$, the corresponding label vector $[y_1, y_2, \ldots, y_n]^T$ as $\mathbf{Y}$, the matrix composed of removed training samples as $\Delta\mathbf{X}$ and the corresponding label vector $[y_{i_1}, y_{i_2}, \ldots, y_{i_k}]^T (i_1, i_2, \ldots, i_k \in \mathcal{R})$, the

formulas above are further rewritten as:

$$\mathbf{w}_{t+1} \leftarrow [(1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{n}\mathbf{X}^T\mathbf{X}]\mathbf{w}_t + \frac{2\eta_t}{n}\mathbf{X}^T\mathbf{Y} \tag{4.35}$$

$$\mathbf{w}^U_{t+1} \leftarrow ((1-\eta_t\lambda)\mathbf{I} - \frac{2\eta_t}{n-\Delta n}(\mathbf{X}^T\mathbf{X} - \Delta\mathbf{X}^T\Delta\mathbf{X}))\mathbf{w}^U_t + \frac{2\eta_t}{n-\Delta n}(\mathbf{X}^T\mathbf{Y} - \Delta\mathbf{X}^T\Delta\mathbf{Y}) \tag{4.36}$$

In the formula above, by further using $\mathbf{M}$ and $\mathbf{N}$ to denote $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{Y}$ respectively, I then conduct eigenvalue decomposition on $\mathbf{X}^T\mathbf{X}$, which results in $\mathbf{M} = \mathbf{Q}\ diag\ (\{c_i\}_{i=1}^n)\ \mathbf{Q}^{-1}$ (where c_i represents the eigenvalues of $\mathbf{M}$). By plugging the decomposed results into Equation (4.35), the following formula can be derived:

$$\begin{aligned}\mathbf{w}_{t+1} = \mathbf{Q}\ diag\ (\{\Pi_{j=1}^t(1-\eta_j\lambda - \frac{2\eta_j}{n}c_i)\}_{i=1}^n)\ \mathbf{Q}^{-1}\mathbf{w}_0 \\ + \mathbf{Q}diag(\sum_{l=1}^{t-1}\eta_l\{\Pi_{j=l+1}^t(1-\eta_j\lambda - \frac{2\eta_j}{n}c_i)\}_{i=1}^n)\ \mathbf{Q}^{-1}\frac{2\mathbf{N}}{n}\end{aligned} \tag{4.37}$$

which avoids the iterative computations of the original GD/SGD update rules. But note that explicitly doing eigenvalue decomposition is very expensive, which is thus infeasible for efficiently computing the exact eigenvalues of $\mathbf{M}' = \mathbf{X}^T\mathbf{X} - \Delta\mathbf{X}^T\Delta\mathbf{X}$ in incremental model update phase. However, by leveraging some theoretical results on incrementally computing the eigenvalues from [132] and regarding $\Delta\mathbf{X}^T\Delta\mathbf{X}$ as small updates on $\mathbf{M}$, I can conclude that the eigenvectors of $\mathbf{M}'$ is approximately the same as that of $\mathbf{M}$ and the eigenvalues of $\mathbf{M}'$ can be estimated as:

$$\mathbf{Q}^{-1}\mathbf{M}'\mathbf{Q} = diag(\{c'_i\}_{i=1}^n) \tag{4.38}$$

where recall that $\mathbf{Q}$ is the matrix of the eigenvectors of $\mathbf{M}$ (see Equation (4.37)). This indicates that $\sum_{i\notin\mathcal{R}}\mathbf{x}_i\mathbf{x}_i^T \approx \mathbf{Q}\ diag\ (\{c'_i\}_{i=1}^n)\ \mathbf{Q}^{-1}$, which is then plugged into Equation

(4.34), i.e.:

$$\begin{aligned}\mathbf{w}_{t+1}^{U} = \mathbf{Q}\ diag\ (\{\Pi_{j=1}^{t}(1-\eta_j\lambda-\frac{2\eta_j}{n-\Delta n}c_i')\}_{i=1}^{n})\ \mathbf{Q}^{-1}\mathbf{w}_0^{U} \\ + \mathbf{Q}diag(\sum_{l=1}^{t-1}\eta_l\{\Pi_{j=l+1}^{t}(1-\eta_j\lambda-\frac{2\eta_j}{n-\Delta n}c_i')\}_{i=1}^{n})\ \mathbf{Q}^{-1}\frac{2\mathbf{N}'}{n-\Delta n}\end{aligned} \tag{4.39}$$

where $\mathbf{N}' = \mathbf{X}^T\mathbf{Y} - \Delta\mathbf{X}^T\Delta\mathbf{Y}$, $\mathbf{Q}$ and $\mathbf{Q}^{-1}$ are computed offline. The time complexity for updating the model parameters in this manner is dominated by the computation of c_i' by using Equation (4.38) and the following computations over each c_i' as Equation (4.37) does. These have time complexities $O(\min\{\Delta n, m\}m^2)$ and $O(Tm)$, respectively. So the total time complexity is $O(\min\{\Delta n, m\}m^2) + O(Tm)$ (recall that there are T iterations in total), which is more efficient than the closed-form solution, whose computational overhead is dominated by the matrix inverse operation, incurring overhead $O(m^3)$. This is much more expensive than one matrix multiplication operation since matrix inverse operation involves multiple matrix multiplication operations, thus leading to higher overhead of closed-form solution than PrIU. In terms of the space overhead, PrIU only requires caching Q, Q^{-1} and all the eigenvalues c_i, which takes space $O(m^2)$. I summarize the time complexity and space complexity of BaseL, PrIU and PrIU-opt for linear regression in Table 4.1 and Table 4.2 respectively.

Theorem 6. ***(Approximation ratio)*** *The approximation of PrIU-opt over the model parameters is bounded by* $O(||\Delta \boldsymbol{X}^T \Delta \boldsymbol{X}||)$

This shows that with small number of removed samples, the approximation ratio should be very small.

4.2.5 PrIU for logistic regression

As the first step to introduce solutions for logistic regression, Equation (4.28) is rewritten as:

$$\mathbf{w}_{t+1}^{LU} \leftarrow [(1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{B_t^U}(\mathbf{C}_t - \Delta\mathbf{C}_t)]\mathbf{w}_t^{LU} + \frac{\eta_t}{B_t^U}(\mathbf{D}_t - \Delta\mathbf{D}_t) \tag{4.40}$$

where $\mathbf{C}_t, \mathbf{D}_t, \Delta\mathbf{C}_t, \Delta\mathbf{D}_t$ are:

$$\mathbf{C}_t = \sum_{i\in\mathcal{B}_t} a_{i,t}\mathbf{x}_i\mathbf{x}_i^T, \Delta\mathbf{C}_t = \sum_{i\in\mathcal{R},i\in\mathcal{B}_t} a_{i,t}\mathbf{x}_i\mathbf{x}_i^T, \mathbf{D}_t = \sum_{i\in\mathcal{B}_t} b_{i,t}y_i\mathbf{x}_i, \Delta\mathbf{D}_t = \sum_{i\in\mathcal{R},i\in\mathcal{B}_t} b_{i,t}y_i\mathbf{x}_i$$

Similar to linear regression, the intermediate results $\mathbf{C}_t$ and $\mathbf{D}_t$ are cached and the dimension of $\mathbf{C}_t$ can be reduced by using SVD before the model update phase, which can happen offline. Suppose after SVD, $\mathbf{C}_t \approx \mathbf{U}_{t,1..z}\mathbf{S}_{t,1..z}\mathbf{V}_{t,1..z}^T$, in which $\mathbf{S}_{t,1..z}$ represents the diagonal matrix with the z largest eigenvalues of $\mathbf{C}_t$ in the diagonal entries and $\mathbf{U}_{t,1..z}$ and $\mathbf{V}_{t,1..z}$ represent the matrix composed of the eigenvectors corresponding to the z largest eigenvalues of $\mathbf{C}_t$. By representing the product $\mathbf{U}_{t,1..z}\mathbf{S}_{t,1..z}$ as $\mathbf{P}_{t,1..z}$, I can derive: $\mathbf{C}_t \approx \mathbf{P}_{t,1..z}\mathbf{V}_{t,1..z}^T$. Note that both $\mathbf{P}_{t,1..z}$ and $\mathbf{V}_{t,1..z}$ are two matrices with dimension $m \times z$. In the end, Equation 4.40 is modified as below for incremental model updates:

$$\mathbf{w}_{t+1}^{LU} \leftarrow [(1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{B_t^U}(\mathbf{P}_{t,1..z}\mathbf{V}_{t,1..z}^T - \Delta\mathbf{C}_t)]\mathbf{w}_t^{LU} + \frac{\eta_t}{B_t^U}(\mathbf{D}_t - \Delta\mathbf{D}_t) \tag{4.41}$$

in which the computation of $\mathbf{P}_{t,1..z}\mathbf{V}_{t,1..z}^T\mathbf{w}_t^{LU}$ and $\Delta\mathbf{C}_t\mathbf{w}_t^{LU}$ become the major overhead, incurring time complexity $O(zm)$ and $O(\Delta Bm)$ respectively for each iteration. Therefore, the total complexity is $O(Tzm)+O(T\Delta Bm)$ if there are T iterations in total. In comparison, the time complexity of retraining from scratch is $O(T((B-\Delta B)m + C_{non}m))$, where C_{non} represents the overhead of the non-linear operations. When $z \ll B$ and $\Delta B \ll B$, PrIU is thus expected to be more efficient than retraining from scratch.

An optimized version of PrIU, PrIU-opt, is come up with in [27] with a series of optimization strategies on minimizing the overhead brought by maintaining provenance. The effect

of those optimization strategies is also empirically demonstrated.

To leverage this approximation, it is essential to cache $\mathbf{P}_{t,1..z}$ and $\mathbf{V}_{t,1..z}$ at each iteration, which requires $O(Trm)$ space in total. Plus, $O(n\lceil\frac{TB}{n}\rceil)$ extra space is necessary to cache the linear coefficients. So the total space complexity will be $O(Trm) + O(n\lceil\frac{TB}{n}\rceil)$.

Theorem 7. ***(Approximation ratio)*** *Similar to Theorem 5, the deviation caused by the SVD approximation will be bounded by $O(\epsilon)$, given the ratio $\frac{||\boldsymbol{P}_{t,1..z}\boldsymbol{V}_{t,1..z}^T||_2}{||\boldsymbol{P}_t\boldsymbol{V}_t^T||_2} \geq 1-\epsilon$. So using Theorem 4, $||E(\boldsymbol{w}_t^{LU} - \boldsymbol{w}_t^{RU})||_2$ is bounded by $O(\frac{\Delta n}{n}\Delta x) + O((\frac{\Delta n}{n})^2) + O((\Delta x)^2) + O(\epsilon)$.*

Optimizations Similar to linear regression, I also proposed an optimized version of PrIU, PrIU-opt for logistic regression when the feature space of the training datasets is small, which depends on one observation that the change in the coefficients $a_{i,t}$ and $b_{i,t}$ from one iteration to the next becomes smaller and smaller as $\mathbf{w}^{(t)}$ converges, thus leading to more and more stable $\mathbf{C}_t$, $\mathbf{D}_t$, $\Delta\mathbf{C}_t$ and $\Delta\mathbf{D}_t$. This indicates that I can stop capturing the linear coefficients $a_{i,t}$ and $b_{i,t}$ at certain iteration t_s, earlier than the convergence. Instead I use the summation of $\mathbf{C}_t$, $\mathbf{D}_t$, $\Delta\mathbf{C}_t$ and $\Delta\mathbf{D}_t$ during the last epoch before the iteration t_s to approximate the matrix $\mathbf{C}_t$, $\mathbf{D}_t$, $\Delta\mathbf{C}_t$ and $\Delta\mathbf{D}_t$ after the iteration t_s, which will remain the same for all iterations $t \geq t_s$, allowing us to avoid their recomputation (I denote the matrices $\mathbf{C}, \mathbf{D}, \Delta\mathbf{C}$ and $\Delta\mathbf{D}$ used after the iteration t_s as $\mathbf{C}_*, \mathbf{D}_*, \Delta\mathbf{C}_*$ and $\Delta\mathbf{D}_*$). Therefore, the update rules for $\mathbf{w}_t^{LU}$ after this approximation becomes:

$$\mathbf{w}_{t+1}^{LU} = \begin{cases} ((1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{B_t^U}(\mathbf{C}_t - \Delta\mathbf{C}_t))\mathbf{w}_t^{LU} + \frac{\eta_t}{B_t^U}(\mathbf{D}_t - \Delta\mathbf{D}_t) & \text{if } t \leq t_s \\ ((1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{n-\Delta n}(\mathbf{C}_* - \Delta\mathbf{C}_*))\mathbf{w}_t^{LU} + \frac{\eta_t}{n-\Delta n}(\mathbf{D}_* - \Delta\mathbf{D}_*) & \text{otherwise} \end{cases} \tag{4.42}$$

In addition, after this approximation, I observe that the update rule after the iteration t_s has the same form as the one for linear regression, motivating us to use the same techniques from PrIU-opt for linear regression, i.e. conducting eigenvalue decomposition over $\mathbf{C}_*$, followed by incrementally updating the eigenvalues given the changes $\Delta\mathbf{C}_t$, thus avoiding recomputations after the iteration t_s. Suppose after applying eigenvalue decomposition on $\mathbf{C}_*$, I have $\mathbf{C}_* = \mathbf{Q}\ diag\ (\{c_i\}_{i=1}^n)\ \mathbf{Q}^{-1}$, which can be followed by the estimations of the

eigenvalues of $\mathbf{C}_* - \Delta\mathbf{C}_*$, i.e.:

$$\mathbf{Q}^{-1}(\mathbf{C}_* - \Delta\mathbf{C}_*)\mathbf{Q} = diag(\{c'_i\}_{i=1}^n) \tag{4.43}$$

Therefore, the iterative computation after the iteration t_s can be avoided, similar to the derivation of Equation (4.39). In the end, Equation (4.42) can be rewritten as:

$$\mathbf{w}_{t+1}^{LU} = \begin{cases} ((1-\eta_t\lambda)\mathbf{I} + \frac{\eta_t}{B_t^U}(\mathbf{C}_t - \Delta\mathbf{C}_t))\mathbf{w}_t^{LU} + \frac{\eta_t}{B_t^U}(\mathbf{D}_t - \Delta\mathbf{D}_t) & \text{if } t \leq t_s \\ \mathbf{Q}\ diag\ (\{\Pi_{j=t_s}^t(1-\eta_j\lambda - \frac{2\eta_j}{n-\Delta n}c'_i)\}_{i=1}^n)\ \mathbf{Q}^{-1}\mathbf{w}_{t_s}^U & \\ +\mathbf{Q}diag(\sum_{l=t_s}^{t-1}\eta_l\{\Pi_{j=l+1}^t(1-\eta_j\lambda - \frac{2\eta_j}{n-\Delta n}c'_i)\}_{i=1}^n)\ \mathbf{Q}^{-1}\frac{2\mathbf{N}'}{n-\Delta n} & \text{otherwise} \end{cases} \tag{4.44}$$

in which $\mathbf{N}' = \frac{\eta_t}{n-\Delta n}(\mathbf{D}_* - \Delta\mathbf{D}_*)$. Then as I analyzed before for the time complexity of linear regression and PrIU for logistic regression, before and after the iteration t_s, the total time complexity is $O(t_s(zm + \Delta Bm))$ and $O(\min\{\Delta n, m\}m^2) + O((T-t_s)m)$ respectively. Thus the total time complexity is $O(t_s(zm + \Delta Bm)) + O(\min\{\Delta n, m\}m^2) + O((T-t_s)m)$. In terms of the space complexity, after the iteration t_s, I only need to keep the eigenvectors of $\mathbf{C}_t$, which requires $O(m^2)$ space. Including the space overhead for the first t_s iterations, the total space complexity is $O(m^2) + O(t_s zm) + O(n\lceil\frac{t_s B}{n}\rceil)$. I summarize the time complexity and space complexity of BaseL, PrIU and PrIU-opt for logistic regression in Table 4.1 and Table 4.2 respectively. The effect of this approximation used in PrIU-opt is theoretically analyzed as below:

Theorem 8. ***(Approximation ratio)*** *Suppose that after the iteration t_s the gradient of the objective function is smaller than δ, then the approximations of PrIU-opt can lead to deviations of the model parameters bounded by $O((T-t_s)\delta) + O(||\Delta\boldsymbol{X}^T\Delta\boldsymbol{X}||)$. By combining the analysis in Theorem 4, $||E(\boldsymbol{w}_t^{LU} - \boldsymbol{w}_t^{RU})||_2$ is bounded by $O(\frac{\Delta n}{n}\Delta x) + O((\frac{\Delta n}{n})^2) + O((\Delta x)^2) + O((T-t_s)\delta) + O(||\Delta\boldsymbol{X}^T\Delta\boldsymbol{X}||)$*

This thus indicates that $\mathbf{w}_t^{LU}$ should be very close to $\mathbf{w}_t^{RU}$.

Discussions

1. **The effect of the mini-batch size** As summarized in Table 4.1, I know that given the same values for all other variables, with larger mini-batch size B, BaseL should be slower while the performance of PrIU and PrIU-opt is free from the mini-batch size, which indicates more significant speed-ups brought by PrIU and PrIU-opt. This is empirically verified in my empirical studies

2. **The effect of the number of iterations T and feature space size m_0** As shown in Table 4.1, for different T and m_0, the speed-ups brought by PrIU and PrIU-opt in comparison to BaseL should be almost remain the same. However, the extra space overhead used to cache the provenance information is proportional to T and m_0 as shown in Table 4.2, implying more memory consumption for larger T and m_0, which is also verified experimentally.

3. **The effect of the approximation in PrIU and PrIU-opt** In PrIU and PrIU-opt, the model parameters are incrementally updated in a faster but approximate manner. I provided theoretical analysis to show the closeness between the approximated results and expected results in Theorem 5-8. I also empirically show that the approximation rate is small even when up to 20% of samples are removed in the model update phase.

4. **Limitations** First of all, my current framework handles linear and logistic models with L2 regularization. My solutions cannot handle L1 regularization since in this case the gradient of the objective function is not continuous, thus invalidating some of the error bound analysis above. In addition, for sparse datasets with large feature space, I can utilize the efficient sparse matrix operations by retraining from scratch, which, however, is not eligible for PrIU since there is no guarantee that $\mathbf{P}_t$ and $\mathbf{V}_t$ after SVD are sparse matrices. Therefore, for sparse training datasets, I will simply use the linearized update rule, i.e. Equation 4.28 directly, without considering the strategies above.

4.2.6 Empirical evaluations

To show the performance advantage of PrIU and PrIU-opt against the approach to retraining from scratch (denoted as BaseL) and other alternative methods, such as INFL [44] and closed-form solutions for linear regression, I provided extensive empirical evaluations on some standard ML benchmark datasets in this section.

4.2.6.1 Experimental setup

Platform. I conduct extensive experiments in Python 3.6 and use PyTorch 1.3.0 [102] for the experiments for dense datasets and scipy 1.3.1 [133] for the experiments for sparse datasets. All experiments were conducted on a Linux server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz and 64GB of main memory.

Datasets. Six datasets were used in my experiments: (1) the UCI SGEMM GPU dataset[2]; (2) the UCI Covtype dataset [3]; (3) the UCI HIGGS dataset [4]; (4) the RCV1 dataset [5] (5) the Kaggle ECG Heartbeat Categorization Dataset[6]; (6) the CIFAR-10 dataset [7], which are referenced as SGEMM, Cov, HIGGS, RCV1, Heartbeat and cifar10 hereafter.

SGEMM has continuous label values, therefore I use it in experiments with *linear regression* while the rest of them have values that are appropriate for classification. Each dataset is partitioned into *training* (90% of the samples) and *validation* (10% of the samples) datasets, the latter used for measuring the accuracy of models trained from the former.

The characteristics of these datasets are listed in Table 4.3, which indicates that RCV1 and cifar10 have extremely large feature space (over 30k model parameters) while other datasets have much fewer parameters (Heartbeat has around 1000 while others have less than 500).

[2] `https://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance`
[3] `https://archive.ics.uci.edu/ml/datasets/covertype`
[4] `https://archive.ics.uci.edu/ml/datasets/HIGGS`
[5] simplified version from https://scikit-learn.org/0.18/datasets/rcv1.html
[6] `https://www.kaggle.com/shayanfazeli/heartbeat`
[7] `https://www.cs.toronto.edu/~kriz/cifar.html`

4.2.6.2 Experiment design

I conduct two sets of experiments, the first of which aims to evaluate the performance of PrIU and PrIU-opt with respect to the deletion of *one subset* of the training samples. I do this over different types of datasets (dense VS sparse, large feature space VS small) with varied configurations (how many samples to be removed, mini-batch size, iteration numbers etc.), and compare against retraining from scratch. The second set of experiments simulate the scenario where users *repetitively* remove different subsets of training samples.

In the first set of experiments I simulate the cleaning scenario. To specify the samples to be removed from the training datasets, I introduce dirty samples, which are a selected subset of samples from the original dataset $\mathcal{T}$ that are modified to incorrect values by rescaling. The resulting dataset is denoted $\mathcal{T}_{\text{dirty}}$, over which the initial model $\mathcal{M}_{\text{init}}$ is constructed. The dirty samples are then removed in the model update phase. The goal is to compare the robustness of PrIU, PrIU-opt and the *influence function* [44] method when dirty data exists. In the experiment I vary the number of erroneous samples generated. The ratio between the erroneous samples and the original training dataset is called the **deletion rate**, and I give it values ranging from 0.0001 (i.e. 0.01%) to 0.2 (i.e. 20%).

In the second set of the experiments, I simulate the scenario in which users debug or interpret models by removing different subsets of samples, necessitating repeated incremental model update operations. I assume that the datasets are very large; to simulate this, I create three synthetic datasets $\mathcal{T}_{\text{cat}}$ by concatenating 4 copies of HIGGS, 20 copies of Cov and 130 copies of Heartbeat such that the total number of training samples is around 40 million, 11 million and 11 million, respectively, which are denoted HIGGS (extended), Cov (extended) and Heartbeat (extended), respectively. In the experiments, ten different subsets are removed and for each of them the deletion rate is about 0.1% of randomly picked samples out of the full training set. The hyperparameters for this set of experiments are listed in Table 4.4.

Baseline. For both $\mathcal{T}_{\text{dirty}}$ and $\mathcal{T}_{\text{cat}}$, I simulate what users (presumably unaware of errors) would do, and train an initial model $\mathcal{M}_{\text{init}}$ using the following standard method: Manually

derive the formula for the gradient of the objective function and then program explicitly the SGD/GD iterations. The erroneous or chosen samples are then removed from $\mathcal{T}_{\text{dirty}}$ or $\mathcal{T}_{\text{cat}}$. For linear regression, I also compare PrIU and PrIU-opt against close-form formula solutions for incremental updates [91, 92, 134, 135], denoted by Closed-form.

Incrementality. To update the model $\mathcal{M}_{\text{init}}$, the straightforward solution is to retrain from the scratch by using the same standard method as before but exclude the removed samples from each mini-batch. I denote this solution by BaseL. In contrast, my approach uses PrIU or PrIU-opt to incrementally update the model. The time taken by BaseL, PrIU or PrIU-opt to produce the updated model is reported in the experiments as the *update time*, and is compared over the two solutions: retraining with BaseL vs. incremental update with PrIU or PrIU-opt.

Note that PrIU/ PrIU-opt uses provenance information collected from the whole training dataset. This phase is *offline* for the PrIU/ PrIU-opt algorithms and is *not* included in their reported running times. In practice, for the first set of experiments (cleaning of erroneous samples) provenance collection is done during the training of $\mathcal{M}_{\text{init}}$ from $\mathcal{T}_{\text{dirty}}$. For the second set of experiments (repeated deletions of subsets for debugging or interpretability) provenance collection is done during an initial training of $\mathcal{M}_{\text{init}}$ from the entire dataset $\mathcal{T}_{\text{cat}}$, which only needs to be done *once* even if many deletions of subsets are performed subsequently.

Since PrIU-opt is the optimized version for datasets with small feature space I only record the update time of PrIU over RCV1 and cifar10, which have very large feature spaces.

Accuracy. I compare the quality of the updated model obtained by BaseL and PrIU/PrIU-opt. The goal is to show that the improvement in update time is not achieved at the expense of accuracy. For experiments with linear regression, I use the *mean squared error (MSE)* over the validation datasets as a measure for accuracy. A lower MSE corresponds to higher accuracy over the validation set. For experiments with binary or multinomial logistic regression, I use the updated model to classify the samples in the validation datasets and report their *validation accuracy*.

Model comparison. I also compare the updated models *structurally* by comparing the vector of updated model parameters obtained via PrIU/PrIU-opt against the ones by using BaseL. This is done in two different ways: 1) Using *distance*, that is, the *L2-norm* of the difference between the two vectors, for both linear and logistic regression, and 2) Using *similarity*, that is, the *cosine* of the angle between the two vectors. The latter is only done for logistic regression since the angle is only relevant for classification techniques. For both linear regression and logistic regression, I also record the changes of the signs and magnitude of individual coordinate of the updated model parameters by PrIU and PrIU-opt compared to the ones obtained by BaseL.

Comparison with influence function. As indicated in Section 2, the *influence function* method in [44] can be extended to handle the removal of multiple training samples by us (details omitted). I denote the resulting method INFL and compare it against PrIU/PrIU-opt in the experiments. I predicted and verified experimentally that this approach produces models with poor validation accuracy since the derivation of INFL relies on the approximation of the Taylor expansion, which can be inaccurate. I also notice that the Taylor expansion used in INFL involves the computation of the Hessian matrix, which is very expensive for datasets with extremely large feature space. So I did not run INFL over RCV1 and cifar10 in the experiments; the comparison between PrIU/PrIU-opt and INFL over other datasets is enough to show the benefits of my approaches.

Effect of the hyperparameters and feature space size As discussed in Section 4.2.4 and Section 4.2.5, the performance of PrIU and PrIU-opt is influenced by the mini-batch size, the number of iterations and the size of the feature space. To explore the effect of the first two parameters for logistic regression, three different combinations of mini-batch size and number of iterations are used over Cov, denoted Cov (small), Cov (large 1) and Cov (large 2) (see Table 4.4). Since the datasets used for logistic regression have different feature space sizes, the performance difference with respect to feature space size is also compared. Since there is only one dataset for linear regression, SGEMM, I extend this dataset by adding 1500 random features for each sample to determine the effect of feature space size.

The extended version of SGEMM is denoted SGEMM (extended) (see Table 4.4). Other hyperparameters used in the experiments are shown in Table 4.4. Note that since erroneous samples exist in the training datasets for the first set of experiments, some values of the learning rate need to be very small to make sure that the convergence can be reached.

In the experiments, I answer the following questions:

(Q1) Do the optimizations used in PrIU-opt compared to PrIU lead to a significant improvement in update time without sacrificing accuracy when the number of features in the training set is small?

(Q2) Do PrIU and PrIU-opt afford significant gains in efficiency compared to BaseL?

(Q3) Are the efficiency gains provided by PrIU and PrIU-opt achieved without sacrificing the accuracy of the updated model?

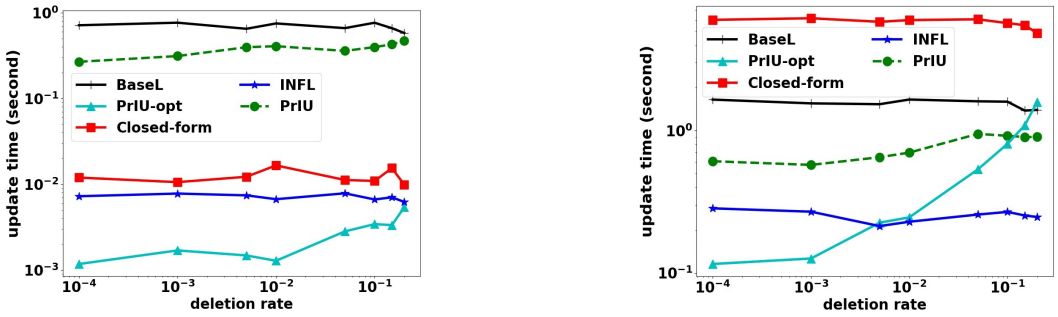
(Q4) Can I experimentally validate the theoretical analysis in Section 4.2.3, i.e. that the updated model derived through the approximations in PrIU and PrIU-opt is very close to the one obtained by BaseL?

(Q5) Does the influence function approach, INFL, provide a competitive alternative to PrIU and PrIU-opt?

(Q6) Can I experimentally show the effect of the hyperparameters, such as mini-batch size and iteration numbers over the performance gains of PrIU and PrIU-opt?

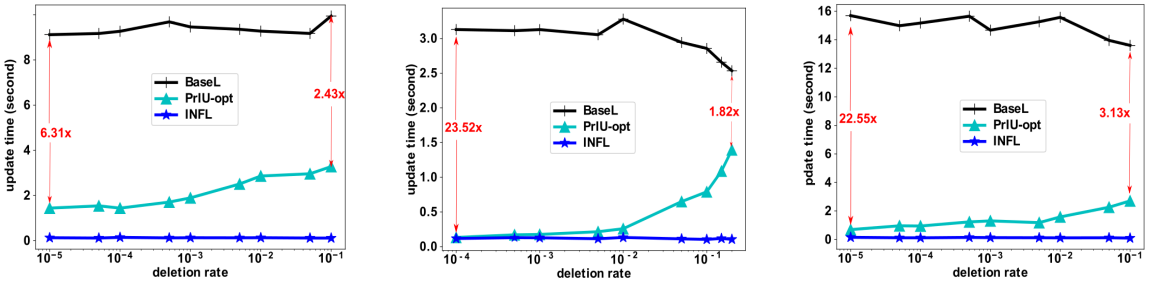
(Q7) Can I experimentally show the effect of the feature space size (i.e. the number of model parameters, which equals to the feature number times the number of classes for multi-nomial logistic regression)?

(Q8) What is the memory overhead of PrIU and PrIU-opt for caching the provenance information?



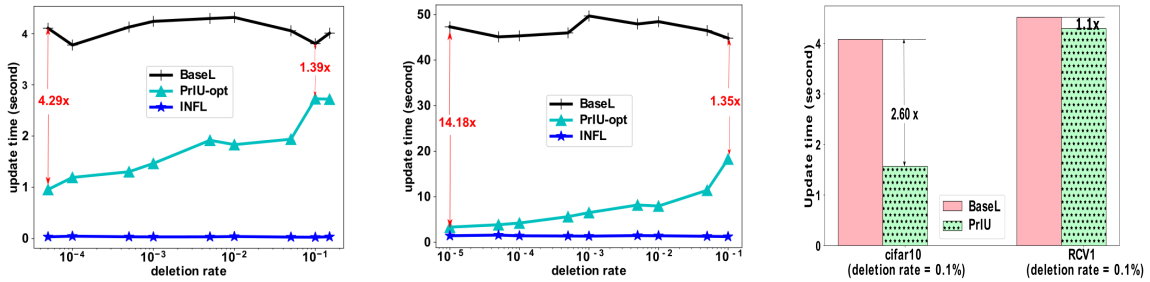
(a) SGEMM (original) (b) SGEMM (extended)

Figure 4.1 – Update time using linear regression



(a) Cov (small) (b) Cov (large 1) (c) Cov (large 2)

Figure 4.2 – Update time using logistic regression over Cov and the hyperparameters from Table 4.4



(a) Heartbeat (b) HIGGS (c) RCV1 and cifar10

Figure 4.3 – Update time using logistic regression

4.2.6.3 Experimental results

I report the results of my experiments in this subsection.

(Q1) I compare the update time of PrIU and PrIU-opt for linear regression using SGEMM (extended) in Figure 4.1b. The results show that the update time of PrIU-opt is significantly better than that of PrIU except when the deletion rate is approaching 20%. I also see from Table 4.6 that PrIU-opt and BaseL yield models that have exactly the same

Table 4.3 – Summary of datasets

name	# features	# classes	# samples
SGEMM	18		241,600
Cov	54	7	581,012
HIGGS	28	2	11,000,000
RCV1	47,236	2	23,149
Heartbeat	188	7	87,553
cifar10	3072	10	50,000

Table 4.4 – Summary of hyperparameters used in the experiments

name	mini-batch size	# of iterations	other hyper-parameters (η, λ)
SGEMM (original)	200	2000	$(5 \times 10^{-3}, 0.1)$
SGEMM (extended)	200	2000	$(5 \times 10^{-3}, 0.1)$
Cov (small)	200	10000	$(1 \times 10^{-4}, 0.001)$
Cov (large 1)	10000	500	$(1 \times 10^{-4}, 0.001)$
Cov (large 2)	10000	3000	$(1 \times 10^{-4}, 0.001)$
HIGGS	2000	20000	$(1 \times 10^{-5}, 0.01)$
Cov (extended)	1000	40000	$(1 \times 10^{-4}, 0.001)$
HIGGS	2000	20000	$(1 \times 10^{-5}, 0.01)$
HIGGS (extended)	2000	60000	$(1 \times 10^{-5}, 0.01)$
Heartbeat	500	5000	$(1 \times 10^{-5}, 0.1)$
Heartbeat (extended)	500	40000	$(1 \times 10^{-5}, 0.1)$
RCV1	500	3000	$(1 \times 10^{-6}, 0.5)$
cifar10	500	1000	$(0.001, 0.1)$

validation accuracy. Therefore, although PrIU-opt uses additional approximations for optimization, they do not hurt the predictive power of the updated models. This shows that the optimization strategies in Sections 4.2.4 and 4.2.5 are worth the design and implementation effort. Consequently, I will only compare PrIU-opt against other approaches except for cifar10 and RCV1 which have extremely large feature spaces.

(Q2) Figures 4.1a-4.1b compare the update time in BaseL and PrIU-opt using linear regression (ignore the INFL lines for the moment), while Figures 4.2-4.3 show the same results for logistic regression for single model update operation. Observe that for both linear and logistic regression, when the deletion rate is small ($<$0.01), PrIU-opt can achieve significant speed-up compared to BaseL: up to two orders of magnitude for linear regression and up to around 23x for logistic regression (for Cov (large 1) and Cov (large 2) with low deletion rate). Even when the feature spaces are extremely large, with deletion rate 0.1%,

Table 4.5 – Memory consumption summary (GB)

Dataset	BaseL	PrIU	PrIU-opt
Cov (small)	0.71	4.30	4.34
Cov (large 1)	0.87	4.02	3.49
Cov (large 2)	1.34	21.0	17.4
HIGGS	5.09	8.40	8.40
SGEMM (original)	2.43	2.45	2.48
SGEMM (extended)	4.94	6.66	5.74
Heartbeat	0.46	6.01	5.69
RCV1	0.28	0.3	-
cifar10	0.79	26.59	-

there is around a 2.6x speed-up for dense datasets (cifar10 in Figure 4.3c) and only 10% for sparse datasets (RCV1 in Figure 4.3c), respectively (similar speed-ups were observed for other small deletion rates). The former shows the effectiveness of the optimization strategies in PrIU over dense datasets with a large feature space while the latter is due to the fact that the optimization strategies for dense datasets were not applied over the sparse ones. Notice that for linear regression, PrIU-opt is always faster than Closed-form. Figure 4.4 shows the results of repetitive model updates; PrIU-opt achieves an order of magnitude speed-up for HIGGS (extended).

(Q3) Table 4.6 (validation accuracy for PrIU and PrIU-opt column) compares the quality of the models obtained by PrIU/PrIU-opt with that of the models obtained by BaseL. For these results I chose the highest deletion rate in the experiments, i.e. 20%. For all the experiments, the validation accuracy (MSE in the case of linear regression) of the updated models obtained by PrIU and PrIU-opt *match exactly* the accuracy of the ones obtained by BaseL. Combined with the answer to **Q2**, I can conclude that *PrIU-opt speeds up the model update time by up to two orders of magnitude without sacrificing any validation accuracy.*

(Q4) I investigate why PrIU-opt has the same validation accuracy as BaseL by measuring the distance and similarity between the updated models computed by PrIU-opt and BaseL. The results are presented in Table 4.6 (again, ignore the columns for INFL). The results indicate that the updated model parameters computed by PrIU-opt are very close to the ones obtained by BaseL since the cosine similarity is almost 1 (see the "similarity"

column) while the L2-dist is very small (see the "distance" column). An even finer-grained analysis, comparing the signs and magnitude of each coordinate in the model parameters updated by PrIU-opt and BaseL shows that there is no sign flipping and only negligible magnitude changes for PrIU-opt compared to BaseL when the deletion rate is small. Even with a large deletion rate of 20% in HIGGS, only 2 out of 58 coordinates flip their signs with small magnitude change.

(Q5) The model update time of INFL is also included in Figures 4.2 and 4.3. Note that it can be up to one order of magnitude better than PrIU-opt, which is expected since using INFL to update the model parameters does not require an iterative computation. However, there is a significant drop in validation accuracy of the updated model derived by INFL compared to BaseL and PrIU-opt (see Table 4.6), which is due to the significantly higher L2-dist (see the "distance" column) and lower cosine similarity (see the "similarity" column) of its updated model compared to the model derived by BaseL. I conclude that PrIU and PrIU-opt produce much better models than INFL yet can still achieve comparable speed-ups.

(Q6) Effect of mini-batch size. The effect of mini-batch size is seen by comparing Cov (large 1) and Cov (small). One observation is that with larger mini-batch size, the maximal speed-up of PrIU-opt is around 23x, while with the smaller mini-batch size it is only about 6x, see Figures 4.2a and 4.2b This confirms the analysis in Section 4.2.5. In the second set of experiments, I used a small mini-batch size for Cov (1000) and Heartbeat (500), resulting in only 4.62x and 3.2x speed-ups by PrIU-opt, respectively (see Figure 4.4).

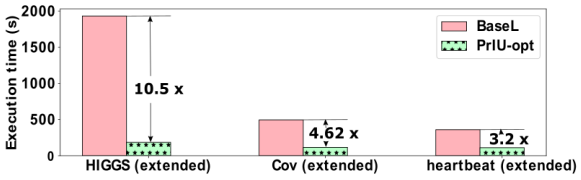


Figure 4.4 – The execution time of repetitively removing 10 different subsets

Effect of number of iterations. A comparison of Cov (large 1) and Cov (large 2),

which have the same mini-batch size but a different number of iterations, can be found in Figures 4.2b and 4.2c. I observe that no matter how many iterations the program runs for, at the same deletion rate PrIU-opt achieves a similar speed-up against BaseL. For example, I have up to around 23x speed-up for small deletion rates and smaller speed-up for higher deletion rates (note the difference in y-axis scale between Figures 4.2b and 4.2c). However, increasing the number of iterations increases the amount of provenance information cached for PrIU-opt, thus requiring more memory. As Table 4.5 indicates, since there are 6x iterations for Cov (large 2) compared to Cov (large 1), roughly 6x memory is needed, confirming the analysis in Section 4.2.5. However for Cov, with a large mini-batch size and 500 iterations, convergence is achieved and I do not observe a difference in validation accuracy between Cov (large 1) and Cov (large 2). Note that according to [136, 137, 138], the theoretical optimal number of passes for logistic regression using mb-SGD (one pass equals to the total number of iterations divided by the number of iterations used for going through the full training set) is quite small. However, for Cov (large 2) the number of passes over the full training set is quite large ($3000/(581012/10000) \approx 60$). Such a high memory usage should therefore not arise in practice.

(Q7) In terms of the update time for experiments over datasets with a comparable mini-batch size but with different feature space sizes (Heartbeat VS HIGGS), I notice that a larger number of model parameters leads to poorer performance by PrIU-opt (compare Figures 4.3a and 4.3b). This is also validated through a second set of experiments in which HIGGS (extended) achieves significant speed-up compared to Heartbeat (extended) (see Figure 4.4). This confirms the analysis in Section 4.2.5, where I show how the asymptotic execution time of PrIU and PrIU-opt depends on the number of the model parameters.

(Q8) Table 4.5 shows that in most cases, both PrIU and PrIU-opt only consume no more than 5x memory compared to BaseL (ignore the number for Cov (large 2) since, as discussed earlier, it is a rare case in practice). However, with a large number of model parameters (like cifar10 and Heartbeat) there is over 10x memory consumption for PrIU and PrIU-opt. How to decrease the memory usage for dense datasets with large feature space is left for

Table 4.6 – Accuracy and similarity comparison between PrIU-opt and INFL with deletion rate 0.2

Dataset	Validation accuracy		distance		similarity	
	BaseL = PrIU-opt	INFL	PrIU-opt	INFL	PrIU-opt	INFL
Cov (small)	48.76%	36.93%	0.184	1.287	0.992	0.624
Cov (large 1)	48.76%	37.99%	0.0016	1.047	1.0	0.738
Cov (large 2)	48.76%	46.38%	0.0003	1.430	1.0	0.471
HIGGS	52.99%	47.99%	0.0004	0.006	0.979	-0.040
Heartbeat	82.78%	74.34%	0.0016	0.583	1.00	0.143
SGEMM (origin)	0.001	0.002	0.027	0.140	-	-
SGEMM (extended)	0.001	0.002	0.029	0.141	-	-

future work.

Discussion. Extensive experiments using linear regression and logistic regression over the datasets above show the feasibility of my approach. PrIU and PrIU-opt can achieve up to two orders of magnitude speed-up for incrementally updating model parameters compared to the baseline, especially for large datasets with a small feature space. This is done without sacrificing the correctness of the results (measured by similarity to the updated model parameters by BaseL) and the prediction performance. The experiments also show that the optimizations used in PrIU-opt give significant performance gains compared to PrIU with only a small loss of accuracy. I observe that INFL is not a good solution because of the poor quality of models produced when more than one sample is removed.

Limitations. My experiments also show the limitations of my solutions. They concern the memory footprint when the feature space or the number of iterations is large (anticipated by several analyses in Section 4.2.5) and the marginal speed-up for large sparse datasets (Also see Section 4.2.5). I shall endeavor to approach these limitations in future work.

4.3 L-BFGS based ML model updates

As the prior section illustrated, PrIU and its optimized version PrIU-opt are only designed for some specific ML models with simple update rules, e.g. linear regression and logistic regression model, which are possibly extended to deal with other *generative additive models*

[28], but very hard to be generalized to more complicated models, e.g. deep neural networks, due to their much more sophisticated update rules than logistic regression. To deal with this issue, I proposed DeltaGrad which can deal with general ML models satisfying strong convexity. In what follows, I take GD as an example to illustrate how DeltaGrad works and discuss how it is extended to SGD later.

4.3.1 DeltaGrad for GD

This subsection begins with descriptions of DeltaGrad for GD, followed by rigorous theoretical analysis for this algorithm.

4.3.1.1 Proposed algorithm

The goal of Deltagrad for GD is to efficiently compute the following *"leave-r-out" gradient formula*:

$$\mathbf{w}_{t+1}^U \leftarrow \mathbf{w}_t^U - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F\left(\mathbf{w}_t^U, \mathbf{z}_i\right) \tag{4.45}$$

after the model parameters $\mathbf{w}_t$ are obtained during the training phase on the full dataset by using the update rule shown in Equation (4.1) and a subset of training samples $\mathcal{R}$ are removed (recall that it is referenced as BaseL for short). This is the baseline method that I discussed in Section 4.2.

In order to efficiently estimate $\mathbf{w}^U$, Equation (4.45) is firstly rewritten as the following form:

$$\mathbf{w}_{t+1}^I = \mathbf{w}_t^I - \frac{\eta_t}{n-r} \left[n \nabla F\left(\mathbf{w}_t^I\right) - \sum_{i \in \mathcal{R}} \nabla F\left(\mathbf{w}_t^I, \mathbf{z}_i\right) \right]. \tag{4.46}$$

where $\mathbf{w}_t^I$ represents the estimated $\mathbf{w}^U$ and $n \nabla F\left(\mathbf{w}_t^I\right) = \sum_{i=1}^n \nabla F\left(\mathbf{w}_t^I, \mathbf{z}_i\right)$. The key idea is to estimate the difference between $\sum_{i=1}^n \nabla F\left(\mathbf{w}_t^I, \mathbf{z}_i\right)$ and $\sum_{i=1}^n \nabla F\left(\mathbf{w}_t, \mathbf{z}_i\right)$ without explicitly computing $\sum_{i=1}^n \nabla F\left(\mathbf{w}_t^I, \mathbf{z}_i\right)$, which is derived by using the well-known Cauchy

mean-value theorem:

$$\nabla F\left(\mathbf{w}_t^I\right) = \nabla F\left(\mathbf{w}_t\right) + \mathbf{H}_t \cdot \left(\mathbf{w}_t^I - \mathbf{w}_t\right) \tag{4.47}$$

in which $\mathbf{H}_t$ is an integrated Hessian, $\mathbf{H}_t = \int_0^1 \mathbf{H}\left(\mathbf{w}_t + x\left(\mathbf{w}_t^I - \mathbf{w}_t\right)\right) dx$.

To avoid explicitly evaluating the hessian matrix, I leverage the classical L-BFGS algorithm (see e.g. [98, 29, 32, 99, 31, 100, 30] and references therein) to effectively and accurately estimate the matrix-vector product $\mathbf{H}_t \cdot \left(\mathbf{w}_t^I - \mathbf{w}_t\right)$. The estimation of this matrix-vector product at the t_{th} iteration utilize Algorithm 8 by leveraging a sequence of past data $\Delta w_{j_1}, \Delta w_{j_2}$, ..., Δw_{j_r} and $\Delta g_{j_1}, \Delta g_{j_2}, \ldots, \Delta g_{j_r}$ evaluated at the iteration j_1, $j_2, \ldots, j_r$, where $\Delta w_j = \mathbf{w}_j^I - \mathbf{w}_j$ and $\Delta g_j = \nabla F(\mathbf{w}_j^I) - \nabla F(\mathbf{w}_j)$.

Algorithm 8: Overview of L-BFGS algorithm

Input : The sequence of the model parameter differences $\Delta W = \{\Delta w_0, \Delta w_1, \ldots, \Delta w_{m_0-1}\}$, the sequence of the gradient differences $\Delta G = \{\Delta g_0, \Delta g_1, \ldots, \Delta g_{m_0-1}\}$, a vector $\mathbf{v}$, history size m_0

Output: Approximate results of $\mathbf{H}(w_{m_0})\mathbf{v}$ at point w_{m_0}, and for some $\mathbf{v}$, such that $\Delta w_i \approx w_i - w_{i-1}$ for all i

1 Compute $\Delta W^T \Delta W$

2 Compute $\Delta W^T \Delta G$, get its diagonal matrix D and its lower triangular submatrix L

3 Compute $\sigma = \Delta g_{m_0-1}^T \Delta w_{m_0-1} / \left(\Delta w_{m_0-1}^T \Delta w_{m_0-1}\right)$

4 Compute the Cholesky factorization for $\sigma \Delta W^T \Delta W + LDL^T$ to get JJ^T

5 Compute $p = \begin{bmatrix} -D^{\frac{1}{2}} & D^{-\frac{1}{2}} L^T \\ \mathbf{0} & J^T \end{bmatrix}^{-1} \begin{bmatrix} D^{\frac{1}{2}} & \mathbf{0} \\ D^{-\frac{1}{2}} L^T & J^T \end{bmatrix}^{-1} \begin{bmatrix} \Delta G^T \mathbf{v} \\ \sigma \Delta W^T \mathbf{v} \end{bmatrix}$

6 **return** $\sigma \mathbf{v} - \begin{bmatrix} \Delta G & \sigma \Delta W \end{bmatrix} p$

As mentioned in Chapter 2, the original L-BFGS algorithm requires explicit evaluations of the gradient formula at each step, which is not ideal for the purpose of incrementally updating model parameters. Therefore, I modified L-BFGS algorithm for my use in DeltaGrad in such a way that the number of iterations in which the gradients are explicitly evaluated is minimized. The algorithmic details of DeltaGrad are shown in Algorithm 9.

In the first j_0 iteration of this algorithm, $\nabla F(\mathbf{w}_t^I)$ is explicitly evaluated and cached. Afterwards, $\nabla F(\mathbf{w}_t^I)$ is only evaluated every T_0 iterations, which is also cached for estimating the gradients for the later iterations. Hence at the iterations t where $t \leq j_0$ or $t - j_0 \mod T_0 = 0$, the gradients $\nabla F(\mathbf{w}_t^I)$ is explicitly evaluated. For all the other iterations,

Algorithm 9: DeltaGrad

Input : The full training set $(\mathbf{X}, \mathbf{Y})$, model parameters cached during the training phase over the full training samples $\{\mathbf{w}_t\}_{t=1}^{T}$ and corresponding gradients $\{\nabla F(\mathbf{w}_t)\}_{t=1}^{T}$, the indices of the removed training samples R, period T_0, total iteration number T, history size m_0, "burn-in" iteration number j_0, learning rate η_t

Output: Updated model parameter $\mathbf{w}_t^I$

Initialize $\mathbf{w}_0^I \leftarrow \mathbf{w}_0$
Initialize an array $\Delta G = []$
Initialize an array $\Delta W = []$
for $t = 0; t < T; t++$ **do**
  if $[((t - j_0) \mod T_0) == 0]$ *or* $t \leq j_0$ **then**
    compute $\nabla F\left(\mathbf{w}_t^I\right)$ exactly
    compute $\nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$ based on the cached gradient $\nabla F\left(\mathbf{w}_t\right)$
    set $\Delta G\left[k\right] = \nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$
    set $\Delta W\left[k\right] = \mathbf{w}_t^I - \mathbf{w}_t$, based on the cached parameters $\mathbf{w}_t$
    $k \leftarrow k + 1$
    compute $\mathbf{w}_{t+1}^I$ by using exact GD update (equation (4.45))
  else
    Pass $\Delta W\left[-m_0 :\right]$, $\Delta G\left[-m_0 :\right]$, the last m_0 elements in ΔW and ΔG, which are from the $j_1^{th}, j_2^{th}, \ldots, j_{m_0}^{th}$ iterations where $j_1 < j_2 < \cdots < j_{m_0}$ depend on t, $\mathbf{v} = \mathbf{w}_t^I - \mathbf{w}_t$, and the history size m_0, to the L-BFGFS Algorithm (see Algorithm 8) to get the approximation of $\mathbf{H}(\mathbf{w}_t)\mathbf{v}$, i.e., $\mathbf{B}_{j_{m_0}}\mathbf{v}$
    Approximate $\nabla F\left(\mathbf{w}_t^I\right) = \nabla F\left(\mathbf{w}_t\right) + \mathbf{B}_{j_{m_0}}\left(\mathbf{w}_t^I - \mathbf{w}_t\right)$
    Compute $\mathbf{w}_{t+1}^I$ by using the "leave-r-out" gradient formula, based on the approximated $\nabla F(\mathbf{w}_t^I)$
  end
end
return $\mathbf{w}_t^I$

$\nabla F(\mathbf{w}_t^I)$ is computed by utilizing Equation (4.47) and the estimation of the product $\mathbf{H}_t \cdot (\mathbf{w}_t^I - \mathbf{w}_t)$ by using Algorithm 8 which utilizes the latest m_0 explicitly evaluated gradients and corresponding parameters, i.e. $\Delta w_{j_1}, \Delta w_{j_2}$, $\ldots$, $\Delta w_{j_{m_0}}$ and $\Delta g_{j_1}, \Delta g_{j_2}, \ldots, \Delta g_{j_{m_0}}$ where $\Delta w_{j_k} = \mathbf{w}_{j_k}^I - \mathbf{w}_{j_k}$, $\Delta g_{j_k} = \nabla F(\mathbf{w}_{j_k}^I) - \nabla F(\mathbf{w}_{j_k})$, $j_k \leq j_0$ or $j_k - j_0 \mod T_0 = 0$, and $t - j_{m_0} < T_0$. Thus the DeltaGrad update is

$$\mathbf{w}_{t+1}^I - \mathbf{w}_t^I = \frac{\eta_t}{n-r} \cdot \begin{cases} \sum_{i \notin \mathcal{R}} \nabla F(\mathbf{w}_t^I, \mathbf{z}_i), \ (t - j_0) \mod T_0 = 0 \text{ or } t \leq j_0 \\ n[\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)] - \sum_{i \in R} \nabla F(\mathbf{w}_t^I, \mathbf{z}_i), \ \text{else} \end{cases} \tag{4.48}$$

in which $\mathbf{B}_{j_{m_0}}$ is the estimation of the integrated hessian matrix $\mathbf{H}_t$ and the product $\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t)$ is computed by Algorithm 8, which is an approximation of the term $\nabla F(\mathbf{w}_t^I) -$

$\nabla F(\mathbf{w}_t)$ according to Equation (4.47). Therefore, $n(\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)) \approx n\nabla F(\mathbf{w}_t^I)$, which also equals to $\sum_{i=1}^{n} \nabla F(\mathbf{w}_t^I, \mathbf{z}_i)$ according to Equation (4.46). By further subtracting the sum of the gradients evaluated on the removed samples, the results will be the approximated gradients on all the remaining samples.

Complexity analysis I will do my complexity analysis assuming that the model is given by a computation graph. Suppose the number of model parameters is p and the time complexity for forward propagation is $f(p)$. Then according to the Baur-Strassen theorem [139], the time complexity of backpropagation in one step will be at most $5f(p)$ and thus the total complexity to compute the derivatives for each training sample is $6f(p)$. Plus, the overhead of computing the product of $\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t)$ is $O(m_0^3) + 6m_0p + p$ according to [32], which means that the total time complexity at the step where the gradients are approximated is $6rf(p) + O(m_0^3) + 6m_0p + p$ (the gradients of r removed/added samples are explicitly evaluated), which is more efficient than explicit computation of the gradients over the full batch (a time complexity of $6(n-r)f(p)$) when $r \ll n$.

Suppose there are T iterations in the training process. Then the running time of BaseL will be $6(n-r)f(p)T$. DeltaGrad evaluates the gradients for the first j_0 iterations and once every T_0 iterations. So its total running time is $6(n-r)f(p) \times \frac{T-j_0}{T_0} + (6rf(p) + O(m_0^3) + 6m_0p + p) \times (1 - \frac{1}{T_0})(T - j_0)$, which is close to $6nf(p) \times \frac{T-j_0}{T_0} + (O(m_0^3) + 6m_0p + p) \times (1 - \frac{1}{T_0})(T - j_0)$ since r is small. Also, when n is large, the overhead of approximate computation, i.e. $(O(m_0^3) + 6m_0p + p)$ should be much smaller than that of explicit computation. Thus *speed-ups of a factor T_0 are expected* when j_0 is far smaller than T.

In terms of the space complexity, DeltaGrad requires storing the model parameters $\mathbf{w}_t$ and the corresponding full gradients $\nabla F(\mathbf{w}_t)$ for all T iterations, which incurs space overhead $O(Tp)$.

4.3.1.2 Theoretical analysis

In this subsection, I provide rigorous theoretical analysis on the closeness between $\mathbf{w}_t^I$ computed by DeltaGrad and $\mathbf{w}_t^U$ computed by BaseL, which shows that the difference of those

two parameters is negligible when only a small portion of the training samples are removed. To start with, some essential assumptions are introduced before the formal descriptions of the main results.

Assumption 1 (Small number of samples removed)**.** *The number of removed samples, r, is far smaller than the total number of training samples, n. There is a small constant $\delta > 0$ such that $r/n \leq \delta$.*

Assumption 2 (Strong convexity and smoothness)**.** *Each $F(\boldsymbol{w}, \boldsymbol{z}_i)$ $(i = 1, 2, \ldots, n)$ is $\mu-$strongly convex and L-smooth with $\mu > 0$, so for any $\boldsymbol{w}_1, \boldsymbol{w}_2$*

$$\left(\nabla F\left(\boldsymbol{w}_1, \boldsymbol{z}_i\right) - \nabla F\left(\boldsymbol{w}_2, \boldsymbol{z}_i\right)\right)^T \left(\boldsymbol{w}_1 - \boldsymbol{w}_2\right) \geq \mu \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|^2,$$

$$\|\nabla F\left(\boldsymbol{w}_1, \boldsymbol{z}_i\right) - \nabla F\left(\boldsymbol{w}_2, \boldsymbol{z}_i\right)\| \leq L \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|.$$

Then $F(\mathbf{w})$ and $F^U(\mathbf{w})$ are L-smooth and μ-strongly convex. Typical choices of η_t are based on the smoothness and strong convexity parameters, so the same choices lead to convergence for both $\mathbf{w}_t$ and $\mathbf{w}_t^U$. For instance, GD over a strongly convex objective with fixed step size $\eta_t = \eta \leq 2/[L + \mu]$ converges geometrically at rate $(L - \mu)/(L + \mu) < 1$. For simplicity, I will use a constant learning rate $\eta_t = \eta \leq 2/[L + \mu]$.

I assume bounded gradients and Lipschitz Hessians, which are standard [140, 141]. The proof may be relaxed to weak growth conditions, see the related works for references.

Assumption 3 (Bounded gradients)**.** *For any model parameter $\boldsymbol{w}$ in the sequence $[\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_t, \ldots]$, the norm of the gradient at every sample is bounded by a constant c_2, i.e. for all i, j:*

$$\|\nabla F\left(\boldsymbol{w}_j, \boldsymbol{z}_i\right)\| \leq c_2.$$

Assumption 4 (Lipschitz Hessian)**.** *The Hessian $\boldsymbol{H}(\boldsymbol{w})$ is Lipschitz continuous. There exists a constant c_0 such that for all $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$,*

$$\|\boldsymbol{H}(\boldsymbol{w}_1) - \boldsymbol{H}(\boldsymbol{w}_2)\| \leq c_0 \|\boldsymbol{w}_1 - \boldsymbol{w}_2\|.$$

An assumption specific to Quasi-Newton methods is the *strong independence* of the weight updates: the smallest singular value of the normalized weight updates is bounded away from zero [142, 143]. This has sometimes been motivated empirically, as the iterates of certain quasi-newton iterations empirically satisfy it [144].

Assumption 5 (Strong independence)**.** *For any sequence,* $\left[\Delta w_{j_1}, \Delta w_{j_2}, \ldots, \Delta w_{j_{m_0}}\right]$*, the matrix of normalized vectors*

$$\Delta W_{j_1, j_2, \ldots, j_{m_0}} = [\Delta w_{j_1}, \Delta w_{j_2}, \ldots, \Delta w_{j_{m_0}}]/s_{j_1, j_{m_0}}$$

where $s_{j_1, j_{m_0}} = \max\left(\|\Delta w_{j_1}\|, \|\Delta w_{j_2}\|, \ldots, \|\Delta w_{j_{m_0}}\|\right)$*, has its minimum singular value* $\sigma_{\min}$ *bounded away from zero. I have* $\sigma_{\min}\left(\Delta W_{j_1, j_2, \ldots, j_{m_0}}\right) \geq c_1$ *where* c_1 *is independent of* $(j_1, j_2, \ldots, j_{m_0})$*.*

Empirically, I find c_1 around 0.2 for the MNIST dataset using my default hyperparameters.

Based on the assumptions above, if hyper-parameters T_0, j_0 are chosen such that

$$A(1-\eta\mu)^{j_0-m+1} d_{0,(m-1)T_0} + \frac{1}{\frac{1}{2}-\frac{r}{n}} AM_1 \frac{r}{n} < \min(\frac{\mu}{2}, (1-\frac{r}{n})\mu - \frac{c_0 M_1 r(n-r)}{2n^2}),$$

e.g. $m_0 = 2$, $T_0 = 5$ and

$$j_0 > \max(\frac{\log(\frac{1}{Ad_{0,5}}[\frac{\mu}{2} - \frac{1}{\frac{1}{2}-\frac{r}{n}} AM_1 \frac{r}{n})]}{\log(1-\eta\mu)}, \frac{\log(\frac{1}{Ad_{0,5}}[(1-\frac{r}{n})\mu - \frac{1}{\frac{1}{2}-\frac{r}{n}} AM_1 \frac{r}{n})]}{\log(1-\eta\mu)}) + m - 1,$$

then I have the following main results on how far $\mathbf{w}_t^I$ is away from the model parameters evaluated on the full training datasets, $\mathbf{w}_t$, and the similarity between the integrated hessian matrix $\mathbf{H}_t$ and $\mathbf{B}_{j_{m_0}}$ at the t_{th} iteration (recall that $\mathbf{H}_t = \int_0^1 \mathbf{H}\left(\mathbf{w}_t + x\left(\mathbf{w}_t^I - \mathbf{w}_t\right)\right) dx$ according to Equation (4.47)).

Theorem 9 (Bound between iterates on full data and incrementally updated ones (all iterations))**.** *Suppose the size of the set of the removed samples* $\mathcal{R}$ *is* r*, then for any* $j_m <$

$t < j_m + T_0 - 1$, $\|\boldsymbol{w}_t^I - \boldsymbol{w}_t\| \leq \frac{1}{\frac{1}{2}-\frac{r}{n}} M_1 \frac{r}{n}$ *and* $\|\boldsymbol{H}_t - \boldsymbol{B}_{j_m}\| \leq \xi_{j_1,j_m}$, *in which* $M_1 = \frac{2c_2}{\mu}$, $\xi_{j_1,j_m} := A d_{j_1,j_m+T_0-1} + A \frac{1}{\frac{1}{2}-\frac{r}{n}} M_1 \frac{r}{n}$ *and* $d_{j_k,j_q} = \max\left(\|\boldsymbol{w}_a - \boldsymbol{w}_b\|\right)_{j_k \leq a \leq b \leq j_q}$.

Theorem (9) suggests that the bound on $\|\mathbf{H}_t - \mathbf{B}_{j_m}\|$ depends on d_{j_1,j_m+T_0-1}, which is analyzed as below.

Lemma 3 (Contraction of the GD iterates)**.** *Recall the definition of* d_{j_k,j_q}*:*

$$d_{j_k,j_q} = \max\left(\|\boldsymbol{w}_a - \boldsymbol{w}_b\|\right)_{j_k \leq a \leq b \leq j_q}.$$

Then $d_{j_k,j_q} \leq d_{j_k-z,j_q-z}$ *for any positive integers* z *and* $d_{j_k,j_q} \leq (1-\mu\eta)^{j_k} d_{0,j_q-j_k}$ *for any* $0 \leq j_k \leq j_q$.

This lemma indicates that d_{j_1,j_m+T_0-1} asymptotically approaches zero when $t \to \infty$, thus implying that $\|\mathbf{H}_t - \mathbf{B}_{j_m}\|$ is asymptotically bounded by $A \frac{1}{\frac{1}{2}-\frac{r}{n}} M_1 \frac{r}{n}$.

In the end, based on those results, the analysis on the approximation ratio of $\mathbf{w}_t^I$ with respect to $\mathbf{w}_t^U$ is provided as below.

Theorem 10 (Convergence rate of DeltaGrad)**.** *For all iterations* t*, the result* $\boldsymbol{w}_t^I$ *of DeltaGrad, Algorithm 9, approximates the correct iteration values* $\boldsymbol{w}_t^U$ *at the rate*

$$\|\boldsymbol{w}_t^U - \boldsymbol{w}_t^I\| = o(\frac{r}{n}).$$

So $\|\boldsymbol{w}_t^U - \boldsymbol{w}_t^I\|$ *is of a lower order than* $\frac{r}{n}$.

This theorem indicates that with proper selections of the hyperparameters j_0, T_0, m_0, $\mathbf{w}_t^I$ is a good approximation to $\mathbf{w}_t^U$, which can be computed very efficiently.

4.3.2 DeltaGrad for SGD

4.3.2.1 Proposed algorithm

When SGD is used to compute the model parameters, by replacing n, $n-r$, $\frac{1}{n-r}\sum_{i \notin \mathcal{R}} \nabla F(\cdot, \mathbf{z}_i)$ and $\nabla F(\cdot)$ with B, B_t^U, $\nabla F(\cdot, \mathcal{B}_t - \mathcal{R})$ and $\nabla F(\cdot, \mathcal{B}_t)$ respectively, Equa-

tion (4.45) and Equation (4.48) become:

$$\mathbf{w}_{t+1}^U \leftarrow \mathbf{w}_t^U - \eta_t \nabla F\left(\mathbf{w}_t^U, \mathcal{B}_t - \mathcal{R}\right) \tag{4.49}$$

$$\begin{aligned}
&\mathbf{w}_{t+1}^I - \mathbf{w}_t^I \\
&= \frac{\eta_t}{|\mathcal{B}_t - \mathcal{R}|} \cdot \begin{cases} \nabla F(\mathbf{w}_t^I, \mathcal{B}_t - \mathcal{R}), \ (t - j_0) \mod T_0 = 0 \text{ or } t \leq j_0 \\ |\mathcal{B}_t|[\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t, \mathcal{B}_t)] - |\mathcal{B}_t \bigcap \mathcal{R}| \nabla F(\mathbf{w}_t^I, \mathcal{B}_t \bigcap \mathcal{R}), \text{ else} \end{cases}
\end{aligned} \tag{4.50}$$

To compute the product of $\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t)$, Algorithm 8 is still utilized with a sequence of past data $\Delta w_{j_1}, \Delta w_{j_2}$, ..., Δw_{j_r} and $\Delta g_{j_1}, \Delta g_{j_2}, \ldots, \Delta g_{j_r}$ evaluated at the iteration j_1, $j_2, \ldots, j_r$, where $\Delta w_j = \mathbf{w}_j^I - \mathbf{w}_j$ and $\Delta g_j = \nabla F(\mathbf{w}_t^I, \mathcal{B}_t) - \nabla F(\mathbf{w}_t, \mathcal{B}_t)$. Then Algorithm 9 is applied to compute $\mathbf{w}_t^I$, which is slightly modified, i.e. all $\nabla F(\cdot)$ are replaced with $\nabla F(\cdot, \mathcal{B}_t)$.

Complexity analysis Recall that in Section 4.3.1, I analyzed the the total time complexity and space complexity of DeltaGrad when GD update rules are used, which are $6nf(p) \times \frac{T-j_0}{T_0} + (O(m_0^3) + 6m_0p + p) \times (1 - \frac{1}{T_0})(T - j_0)$ and $O(Tp)$ respectively. In the case of SGD update rules, by replacing n ith B, the time complexity of DeltaGrad becomes $6Bf(p) \times \frac{T-j_0}{T_0} + (O(m_0^3) + 6m_0p + p) \times (1 - \frac{1}{T_0})(T - j_0)$ while the space complexity is still $O(Tp)$. Note that in the context of binary logistic regression, the number of model parameters p equal to the feature number m_0. Plus, recall that $6Bf(p)$ represents the overhead to compute the explicit gradients over the full mini-batch, which equals to $O((B - \Delta B)m + C_{non}m)$ for the update rule of the binary logistic regression. As a result, the time complexity of DeltaGrad for binary logistic regression is reformulated as $O((B - \Delta B)m + C_{non}m) \times \frac{T-j_0}{T_0} + (O(m_0^3) + 6m_0m + m) \times (1 - \frac{1}{T_0})(T - j_0)$, which is compared against the performance of PrIU and PrIU-opt in Table 4.7. This result shows that with larger m_0, DeltaGrad saves more memory in comparison to PrIU and PrIU-opt since it has lower space complexity, in which case, PrIU is preferred to PrIU-optsince the latter one incurs memory overhead quadratic to the feature number m_0. In terms of the time complex-

ity, with larger m_0 than B, the empirical choice of z and T_0 is $B/5$ and 5 in my experiments for PrIU and DeltaGrad respectively, which can achieve adequate trade-offs between the approximation rate and efficiency, in which configurations, the dominate time overhead of PrIU and DeltaGrad become $O(Tzm) = O(\frac{BTm}{5})$ and $O((B-\Delta B)m+C_{non}m)\times\frac{T-j_0}{T_0} \approx O(\frac{BTm}{5})$. But since PrIU incurs more memory overhead than DeltaGrad, I should expect better performance of DeltaGrad than PrIU for datasets with large feature space, which is empirically verified in my experiments.

	PrIU	PrIU-opt	DeltaGrad
Space complexity	$O(Tzm) + O(n\lceil\frac{TB}{n}\rceil)$	$O(m^2) + O(t_s zm) + O(n\lceil\frac{t_s B}{n}\rceil)$	$O(Tm)$
Time complexity	$O(Tzm) + O(T\Delta Bm)$	$O(t_s(zm + \Delta Bm)) + O(\min\{\Delta n, m\}m^2) + O((T - t_s)m)$	$O((B-\Delta B)m+C_{non}m)\times\frac{T-j_0}{T_0} + (O(m_0^3) + 6m_0 m + m)\times(1 - \frac{1}{T_0})(T - j_0)$

Table 4.7 – Complexity comparison between PrIU, PrIU-opt and DeltaGrad for binary logistic regression. The complexity expressions of PrIU and PrIU-opt are copied from Table 4.1-4.2

4.3.2.2 Theoretical results

Theorem 11 (SGD bound for DeltaGrad)**.** *With probability at least*

$$1 - T\cdot[2p\exp(-\sqrt{B}\log(2p)/[4+\frac{2}{3}\left(\frac{\log(2p)}{B}^2\right)^{1/4}]) + (p+1)\exp(-\frac{\log(p+1)\sqrt{B}}{4+\frac{2}{3}\left(\frac{\log(p+1)}{B}^2\right)^{1/4}}) + 2\exp(-2\sqrt{B})],$$

the result $\boldsymbol{w}_t^I$ computed by Equation (4.50) *approximates the correct iteration values ${\boldsymbol{w}^{U,S}}_t$ at the rate*

$$\|{\boldsymbol{w}^{U,S}}_t - {\boldsymbol{w}^{I,S}}_t\| = o\left(\frac{r}{n} + \frac{1}{B^{\frac{1}{4}}}\right).$$

Thus, when B is large, and when r/n is small, my algorithm accurately approximates the correct iteration values.

Discussions As a by-product, DeltaGrad can also handle the incremental additions for

general ML models satisfying strong convexity. Specifically, suppose the set of newly added training sample is $\mathcal{A}$, then at each iteration, I randomly sample a mini-batch $\mathcal{A}_t$ from $\mathcal{A}$, and combine it with the mini-batch $\mathcal{B}_t$ (used for previous training phase on the original datasets) to compose a new mini-batch. Therefore, the update rule of DeltaGrad in Equation (4.50) can be modified as:

$$\begin{aligned}&\mathbf{w}_{t+1}^I - \mathbf{w}_t^I \\ &= \frac{\eta_t}{|\mathcal{B}_t| + |\mathcal{A}_t|} \cdot \begin{cases} |\mathcal{B}_t|\nabla F(\mathbf{w}_t^I, \mathcal{B}_t) + |\mathcal{A}_t|\nabla F(\mathbf{w}_t^I, \mathcal{A}_t), \ (t - j_0) \mod T_0 = 0 \text{ or } t \leq j_0 \\ |\mathcal{B}_t|[\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t, \mathcal{B}_t)] + |\mathcal{A}_t|\nabla F(\mathbf{w}_t^I, \mathcal{A}_t), \text{ else} \end{cases}\end{aligned} \tag{4.51}$$

in which $\mathcal{A}_t$ represents the randomly selected samples from the newly added sample set at the t_{th} iteration. Based on the above analysis, in the presence of both the deletions and additions of a small amount of training samples, Equation (4.50) and Equation (4.51) can be combined as follows:

$$\begin{aligned}&\mathbf{w}_{t+1}^I - \mathbf{w}_t^I = \frac{\eta_t}{|\mathcal{B}_t - \mathcal{R}| + |\mathcal{A}_t|} \\ &\cdot \begin{cases} (|\mathcal{B}_t - \mathcal{R}| + |\mathcal{A}_t|)\nabla F(\mathbf{w}_t^I, (\mathcal{B}_t - \mathcal{R}) \bigcup \mathcal{A}_t), \ (t - j_0) \mod T_0 = 0 \text{ or } t \leq j_0 \\ |\mathcal{B}_t|[\mathbf{B}_{j_{m_0}}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t, \mathcal{B}_t)] - |\mathcal{B}_t \bigcap \mathcal{R}|\nabla F(\mathbf{w}_t^I, \mathcal{R}) \\ + |\mathcal{A}_t|\nabla F(\mathbf{w}_t^I, \mathcal{A}_t) \end{cases} \text{, else}\end{aligned} \tag{4.52}$$

which can then guide us to generalize Algorithm 9 by taking both the additions and deletions of small amount of training samples into consideration when SGD is used for training (see Algorithm 10).

4.3.3 Extension to online deletions/additions

I also extended DeltaGrad to the applications where online deletions/additions are necessary, for which the history information used in DeltaGrad is updated after each deletion/addition request is over, prepared for the following deletion/addition requests. The details of the

Algorithm 10: DeltaGrad(deletions + additions for SGD)

Input : A training set $\mathcal{Z}$, a set of added training samples, $\mathcal{A}$, a set of deleted training samples, $\mathcal{R}$, total number of the SGD iterations, T, the model parameters and gradients cached before $\mathcal{Z}$ is updated, $\{\mathbf{w}_t\}_{t=1}^{T}$ and $\{\nabla F_{\mathbf{w}}(\mathbf{w}_t, \mathcal{B}_t)\}_{t=1}^{T}$, period T_0, history size m_0, "burn-in" iteration number j_0

Output: Updated model parameter $\mathbf{w}_T^I$

Initialize $\mathbf{w}_0^I \leftarrow \mathbf{w}_0$, $\Delta G = []$, $\Delta W = []$
for $t = 0; t < T; t++$ **do**
    randomly sample a mini-batch, $\mathcal{A}_t$, from $\mathcal{A}$
    if $[((t - j_0) \mod T_0) == 0]$ *or* $t \leq j_0$ **then**
        explicitly compute $\nabla_{\mathbf{w}} F\left(\mathbf{w}_t^I; \mathcal{B}_t\right)$
        compute $\nabla_{\mathbf{w}} F\left(\mathbf{w}_t^I; (\mathcal{B}_t - \mathcal{R}) \cup \mathcal{A}_t\right)$
        set $\Delta G[r] = \nabla F\left(\mathbf{w}_t^I; \mathcal{B}_t\right) - \nabla F(\mathbf{w}_t; \mathcal{B}_t)$, $\Delta W[r] = \mathbf{w}_t^I - \mathbf{w}_t$
        $r \leftarrow r + 1$
    else
        pass the last m_0 elements in ΔW and ΔG, and $\mathbf{v} = \mathbf{w}_t^I - \mathbf{w}_t$ to the L-BFGFS Algorithm to calculate the product, $\mathbf{B}_t\mathbf{v}$
        Pass $\Delta W[-m_0:]$, $\Delta G[-m_0:]$, the last m_0 elements in ΔW and ΔG, which are from the $j_1^{th}, j_2^{th}, \ldots, j_{m_0}^{th}$ iterations where $j_1 < j_2 < \cdots < j_{m_0}$ depend on t, $\mathbf{v} = \mathbf{w}_t^I - \mathbf{w}_t$, and the history size m_0, to the L-BFGFS Algorithm (see Algorithm 8) to get the approximate Hessian-vector product, $\mathbf{B}_{j_{m_0}}\mathbf{v}$
        Approximate $\nabla F\left(\mathbf{w}_t^I\right) = \nabla F(\mathbf{w}_t) + \mathbf{B}_{j_{m_0}}\left(\mathbf{w}_t^I - \mathbf{w}_t\right)$
        Compute $\mathbf{w}_{t+1}^I$ by using the "leave-r-out" gradient formula, based on the approximated $\nabla F(\mathbf{w}_t^I)$
        approximate $\nabla_{\mathbf{w}} F\left(\mathbf{w}_t^I, \mathcal{B}_t\right)$ and compute $\nabla_{\mathbf{w}} F\left(\mathbf{w}_t^I; (\mathcal{B}_t - \mathcal{R}) \cup \mathcal{A}_t\right)$ by utilizing Equation (4.52)
    end
end
update $\mathbf{w}_t^I$ to $\mathbf{w}_{t+1}^I$ with $\nabla_{\mathbf{w}} F\left(\mathbf{w}_t^I; (\mathcal{B}_t - \mathcal{R}) \cup \mathcal{A}_t\right)$
return $\mathbf{w}_T^I$

online version of DeltaGrad are presented in Algorithm 11 with modifications on Algorithm 9 highlighted.

Algorithm 11: DeltaGrad (online deletion/addition)

Input : The full training set $(\mathbf{X}, \mathbf{Y})$, model parameters cached during the training phase for the full training samples $\{\mathbf{w}_t\}_{t=1}^{T}$ and corresponding gradients $\{\nabla F(\mathbf{w}_t)\}_{t=1}^{T}$, the index of the removed training sample or the added training sample i_r, period T_0, total iteration number T, history size m_0, warmup iteration number j_0, learning rate η

Output: Updated model parameter $\mathbf{w}_t^I$

Initialize $\mathbf{w}_0^I \leftarrow \mathbf{w}_0$
Initialize an array $\Delta G = []$
Initialize an array $\Delta W = []$
for $t = 0; t < T; t++$ **do**
  if $[((t - j_0) \mod T_0) == 0]$ *or* $t \leq j_0$ **then**
    compute $\nabla F\left(\mathbf{w}_t^I\right)$ exactly
    compute $\nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$ based on the cached gradient $\nabla F\left(\mathbf{w}_t\right)$
    set $\Delta G\left[k\right] = \nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$
    set $\Delta W\left[k\right] = \mathbf{w}_t^I - \mathbf{w}_t$, based on the cached parameters $\mathbf{w}_t$
    $k \leftarrow k + 1$
    compute $\mathbf{w}_{t+1}^I$ by using exact GD update (equation (4.45))
    $\mathbf{w}_t \leftarrow \mathbf{w}_t^I$
    $\nabla F(\mathbf{w}_t) \leftarrow \nabla F(\mathbf{w}_t^I)$
  else
    Pass $\Delta W\left[-m:\right]$, $\Delta G\left[-m:\right]$, the last m_0 elements in ΔW and ΔG, which are from the $j_1^{th}, j_2^{th}, \ldots, j_m^{th}$ iterations where $j_1 < j_2 < \cdots < j_m$ depend on t, $\mathbf{v} = \mathbf{w}_t^I - \mathbf{w}_t$, and the history size m_0, to the L-BFGFS Algorithm (See Supplement) to get the approximation of $\mathbf{H}(\mathbf{w}_t)\mathbf{v}$, i.e., $\mathbf{B}_{j_m}\mathbf{v}$
    Approximate $\nabla F\left(\mathbf{w}_t^I\right) = \nabla F\left(\mathbf{w}_t\right) + \mathbf{B}_{j_m}\left(\mathbf{w}_t^I - \mathbf{w}_t\right)$
    Compute $\mathbf{w}_{t+1}^I$ by using the "leave-1-out" gradient formula, based on the approximated $\nabla F(\mathbf{w}_t^I)$
    $\mathbf{w}_t \leftarrow \mathbf{w}_t^I$
    $\nabla F(\mathbf{w}_t) \leftarrow \frac{\eta}{n-1}[n(\mathbf{B}_{j_m}(\mathbf{w}_t^I - \mathbf{w}_t) + \nabla F(\mathbf{w}_t)) - \nabla F(\mathbf{w}_t, \mathbf{z}_{i_r})]$
  end
end
return $\mathbf{w}_t^I$

As shown in Algorithm 11, the history information used at each deletion request, i.e. $\mathbf{w}_t$ and $\nabla F(\mathbf{w}_t)$, is updated at prior deletion requests, which may lead to gradual deviations of $\mathbf{w}_t^I$ from $\mathbf{w}_t^U$ with more and more deletion requests are processed. However, with a reasonable assumption that the total number of removed samples is still a small portion of the entire training dataset, I can provide rigorous analysis on the approximation rate of the online version DeltaGrad. In this analysis, I only assume the occurrence of online deletions, which,

however, is pretty straightforward to be generalized to online additions and the mixture of online additions and online deletions. By using $\mathbf{w}_t^I(r)$ and $\mathbf{w}_t^U(r)$ to denote the resulting model parameters updated by DeltaGrad and BaseL after the r_{th} deletion requests at the t_{th} iteration respectively, I have the following results for the online version of DeltaGrad.

Theorem 12 (Convergence rate of DeltaGrad (online deletion))**.** *At the r_{th} deletion request, for all iterations t, the result $\boldsymbol{w}_t^I(r)$ of DeltaGrad, Algorithm 11, approximates the correct iteration values $\boldsymbol{w}_t^U(r)$ at the rate*

$$\|\boldsymbol{w}_t^U(r) - \boldsymbol{w}_t^I(r)\| = o(\frac{r}{n}).$$

So $\|\boldsymbol{w}_t^U(r) - \boldsymbol{w}_t^I(r)\|$ is of a lower order than $\frac{r}{n}$.

This suggests that $\mathbf{w}_t^I(r)$ and $\mathbf{w}_t^U(r)$ are still pretty close in the online addition/deletion scenario.

4.3.4 Extension to DNNs

For the original version of the L-BFGS algorithm, strong convexity for the objective function is essential. In this subsection, I present my extension of DeltaGrad to non-strongly convex, non-smooth objectives.

To deal with non-strongly convex objectives, I assume that convexity holds in some local regions. When constructing the arrays ΔG and ΔW, only the model parameters and their gradients where local convexity holds are used.

For local non-smoothness, I found that even a small distance between $\mathbf{w}_t$ and $\mathbf{w}_t^I$ can make the estimated gradient $\nabla F(\mathbf{w}_t^I)$ drift far away from $\nabla F(\mathbf{w}_t)$. To deal with this, I explicitly check if the norm of $\mathbf{B}_{j_m}(\mathbf{w}_t - \mathbf{w}_t^I)$ (which equals to $\nabla F(\mathbf{w}_t^I) - \nabla F(\mathbf{w}_t)$) is larger than the norm of $L(\mathbf{w}_t - \mathbf{w}_t^I)$ for a constant L. In my experiments, L is configured as 1. The details of the modifications above are highlighted in Algorithm 12. Since whenever local convexity or smoothness is violated at certain iteration t, I have to explicitly evaluate the gradients on the current mini-batch, which, in the worst case, will force the explicit gradient

evaluations at all iterations, thus reduced to BaseL,

4.3.5 Empirical studies

4.3.5.1 Experimental setup

Datasets. I used four datasets for evaluation: MNIST [145], covtype [146], HIGGS [147] and RCV1 [148] [8] . MNIST contains 60,000 images as the training dataset and 10,000 images as the test dataset; each image has 28×28 features (pixels), containing one digit from 0 to 9. The covtype dataset consists of 581,012 samples with 54 features, each of which may come from one of the seven forest cover types; as a test dataset, I randomly picked 10% of the data. HIGGS is a dataset produced by Monte Carlo simulations for binary classification, containing 21 features with 11,000,000 samples in total; 500,000 samples are used as the test dataset. RCV1 is a corpus dataset; I use its binary version which consists of 679,641 samples and 47,236 features, of which the first 20,242 samples are used for training.

Machine configuration. All experiments are run over a GPU machine with one Intel(R) Core(TM) i9-9920X CPU with 128 GB DRAM and 4 GeForce 2080 Titan RTX GPUs (each GPU has 10 GB DRAM). I implemented DeltaGrad with PyTorch 1.3 and used one GPU for accelerating the tensor computations.

Deletion/Addition benchmark. I run regularized logistic regression over the four datasets with L2 norm coefficient 0.005, fixed learning rate 0.1. The mini-batch sizes for RCV1 and other three datasets are 16384 and 10200 respectively (Recall that RCV1 only has around 20k training samples). I also evaluated my approach over a two-layer neural network with 300 hidden ReLU neurons over MNIST. There L2 regularization with rate 0.001 is added along with decaying learning rate (first 10 iterations with learning rate 0.2 while the rest of iterations with learning rate 0.1) and batch size equal to the training dataset size (i.e. deterministic gradient descent is used). Note that there are no strong convexity and smoothness guarantees for DNN models. Thus some modifications are made over Algorithm 9 (see Algorithm 12).

[8] I used its binary version from LIBSVM:

I evaluate two cases of addition/deletion: *batch* and *online*. Multiple samples are grouped together for addition and deletion in the former, while samples are removed one after another in the latter. Algorithm 9 are slightly modified to fit the online deletion/addition cases (see Algorithm 11 in the Appendix). To simulate deleting training samples, $\mathbf{w}^*$ is evaluated over the full training dataset of n samples, which is followed by the random removal of r samples and evaluation over the remaining $n - r$ samples using BaseL or DeltaGrad. To simulate adding training samples, r samples are deleted first. After $\mathbf{w}^*$ is evaluated over the remaining $n - r$ samples, the r samples are added back to the training set for updating the model. The ratio of r to the total number of training samples n is called the *Delete rate* and *Add rate* for the two scenarios, respectively. I also provided other experiments in this section to compare the performance of DeltaGrad with that of state-of-the-art work, and study the effect of mini-batch sizes and hyper-parameters of DeltaGrad.

Throughout the experiments, the running time of BaseL and DeltaGrad to update the model parameters is recorded. To show the difference between $\mathbf{w}^{U*}$ (the output of BaseL, and the correct model parameters after deletion or addition) and $\mathbf{w}^{I*}$ (the output of DeltaGrad), I compute the ℓ_2-norm or distance $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$. For comparison, and justify the theory in Section 4.3.1.2 and Section 4.3.2.2, $\|\mathbf{w}^* - \mathbf{w}^{U*}\|$ is also recorded ($\mathbf{w}^*$ are the parameters trained over the full training data). Given the same set of added or deleted samples, the experiments are repeated 10 times, with different minibatch randomness each time. After the model updates, $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ are evaluated over the test dataset and their prediction performance is reported.

Hyperparameter setup. I set T_0 (the period of explicit gradient updates) and j_0 (the length of the inital "burn-in") as follows. For regularized logistic regression, I set $T_0 = 10, j_0 = 10$ for RCV1, $T_0 = 5, j_0 = 10$ for MNIST and covtype, and $T_0 = 3, j_0 = 300$ for HIGGS. For the 2-layer DNN, $T_0 = 2$ is even smaller and the first quarter of the iterations are used as "burn-in". The history size m_0 is 2 for all experiments.

4.3.6 Experimental results

4.3.6.1 Batch addition/deletion.

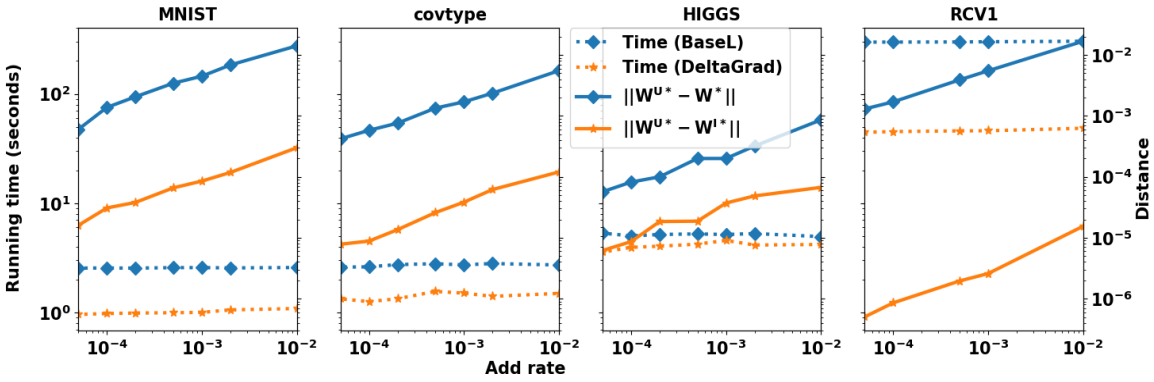


Figure 4.5 – Running time and distance with varied add rate

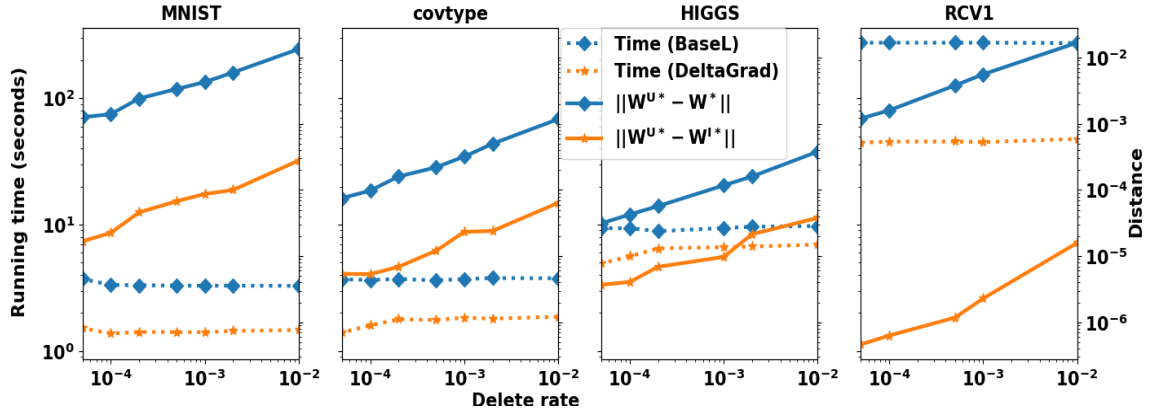


Figure 4.6 – Running time and distance with varied delete rate

To test the robustness and efficiency of DeltaGrad in batch deletion, I vary the *Delete* and *Add rate* from 0 to 0.01. Figures 4.5 and 4.6 show the running time of BaseL and DeltaGrad (blue and red dotted lines, resp.) and the two distances, $\|\mathbf{w}^{U*} - \mathbf{w}^*\|$ and $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ (blue and red solid lines, resp.) over the four datasets using regularized logistic regression. The results on the use of 2-layer DNN over MNIST are presented in Figure 4.7, which is denoted by MNIST^n.

The running time of BaseL and DeltaGrad is almost constant regardless of the delete or

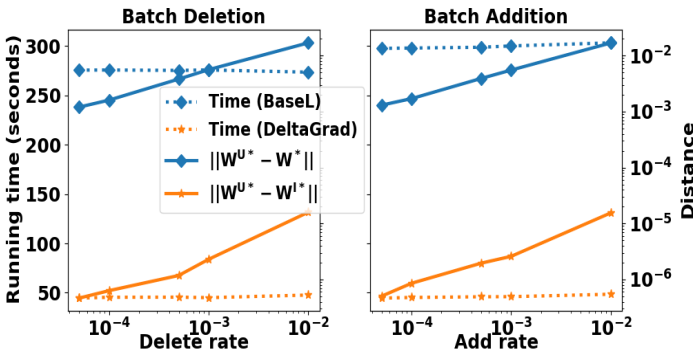


Figure 4.7 – Running time and distance with varied delete rate/add rate for MNIST^n

add rate, confirming the time complexity analysis of DeltaGrad in Section 4.3.1 and Section 4.3.2. The theoretical running time is free of the number of removed samples r, when r is small. For any given delete/add rate, DeltaGrad achieves significant speed-ups (up to 2.6x for MNIST, 2x for covtype, 1.6x for HIGGS, 6.5x for RCV1) compared to BaseL. On the other hand, the distance between $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ is quite small; it is less than 0.0001 even when up to 1% of samples are removed or added. When the delete or add rate is close to 0, $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ is of magnitude 10^{-6} (10^{-8} for RCV1), indicating that the approximation brought by $\mathbf{w}^{I*}$ is negligible. Also, $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ is at least one order of magnitude smaller than $\|\mathbf{w}^{U*} - \mathbf{w}^{*}\|$, confirming my theoretical analysis comparing the bound of $\|\mathbf{w}^{U*} - \mathbf{w}^{I*}\|$ to that of $\|\mathbf{w}^{U*} - \mathbf{w}^{*}\|$.

To investigate whether the tiny difference between $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ will lead to any difference in prediction behavior, the prediction accuracy using $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ is presented in Table 4.8. Due to space limitations, only results on a very small (0.005%) and the largest (1%) add/delete rates are presented. Due to the randomness in SGD, the standard deviation for the prediction accuracy is also presented. In most cases, the models produced by BaseL and DeltaGrad end up with effectively the same prediction power. There are a few cases where the prediction results of $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ are not exactly the same (e.g. Add (1%) over MNIST), their confidence intervals overlap, so that statistically $\mathbf{w}^{U*}$ and $\mathbf{w}^{I*}$ provide the same prediction results.

For the 2-layer net model where strong convexity does not hold, see the last sub-figures in

Figure 4.5 and 4.6. The figure shows that DeltaGrad achieves about 1.4x speedup compared to BaseL while maintaining a relatively small difference between $\mathbf{w}^{I*}$ and $\mathbf{w}^{U*}$. This suggests that it may be possible to extend my analysis for DeltaGrad beyond strong convexity; this is left for future work.

4.3.6.2 Online addition/deletion.

To simulate deletion and addition requests over the training data continuously in an on-line setting, 100 random selected samples are added or deleted sequentially. Each triggers model updates by either BaseL or DeltaGrad. The running time comparison between the two approaches in this experiment is presented in Figure 4.8, which shows that DeltaGrad is about 2.5x, 2x, 1.8x and 6.5x faster than BaseL on MNIST, covtype, HIGGS and RCV1 respectively. The accuracy comparison is shown in Table 4.9. There is essentially no prediction performance difference between $\mathbf{w}^{U*}$ and $\mathbf{w}^{*}$.

Discussion. By comparing the speed-ups brought by DeltaGrad and the choice of T_0, I found that the theoretical speed-ups are not fully achieved. One reason is that in the approximate L-BFGS computation, a series of small matrix multiplications are involved. Their computation on GPU vs CPU cannot bring about very significant speed-ups compared to the larger matrix operations[9], which indicates that the overhead of L-BFGS is non-negligible compared to gradient computation. Besides, although r is far smaller than n, to compute the gradients over the r samples, other overhead becomes more significant: copying data from CPU DRAM to GPU DRAM, the time to launch the kernel on GPU, etc. This leads to non-negligible explicit gradient computation cost over the r samples. It would be interesting to explore how to adjust DeltaGrad to fully utilize the computation power of GPU in the future.

[9]See the matrix computation benchmark on GPU with varied matrix sizes: https://developer.nvidia.com/cublas

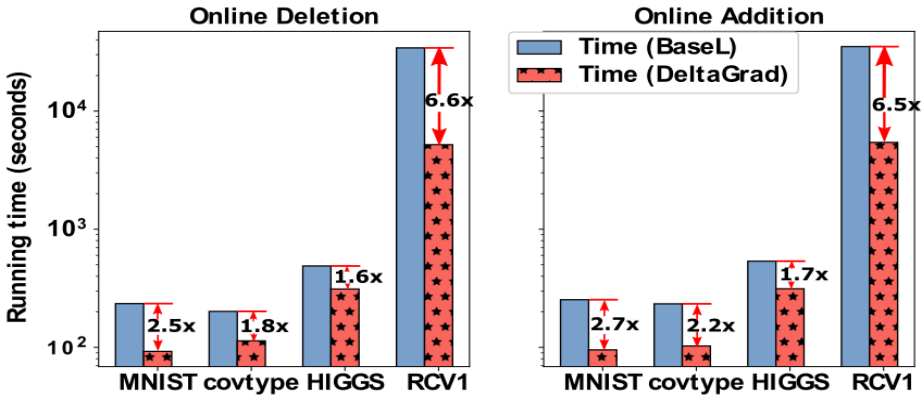


Figure 4.8 – Running time comparison of BaseL and DeltaGrad with 100 continuous deletions/addition

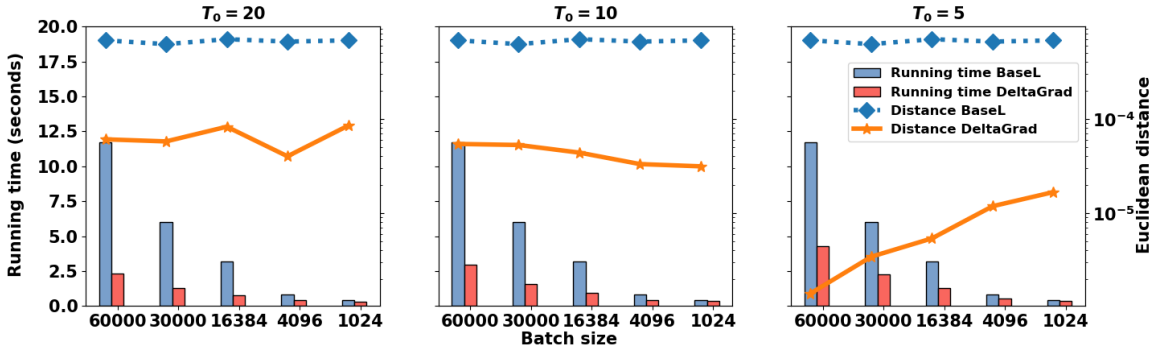


Figure 4.9 – Running time and distance comparison with varying mini-batch size under fixed $j_0 = 10$ and varying T_0 ($T_0 = 20$ VS $T_0 = 10$ VS $T_0 = 5$)

4.3.6.3 Influence of hyper-parameters on performance

To begin with, the influence of different hyper-parameters used in SGD and DeltaGrad is explored. I delete one sample from the training set of MNIST by running regularized logistic regression with the same learning rate and regularization rate as in Section 4.3.5.1 and varying mini-batch sizes (1024 - 60000), T_0 ($T_0 = 20$, 10, 5) and j_0 ($j_0 = 5$, 10, 50). The experimental results are presented in Figure 4.9-4.10. For different mini-batch sizes, I also used different epoch numbers to make sure that the total number of running iterations/steps in SGD are roughly the same. In what follows, I analyze how the mini-batch size, the hyper-parameters T_0 and j_0 influence the performance, thus providing some hints on how to choose

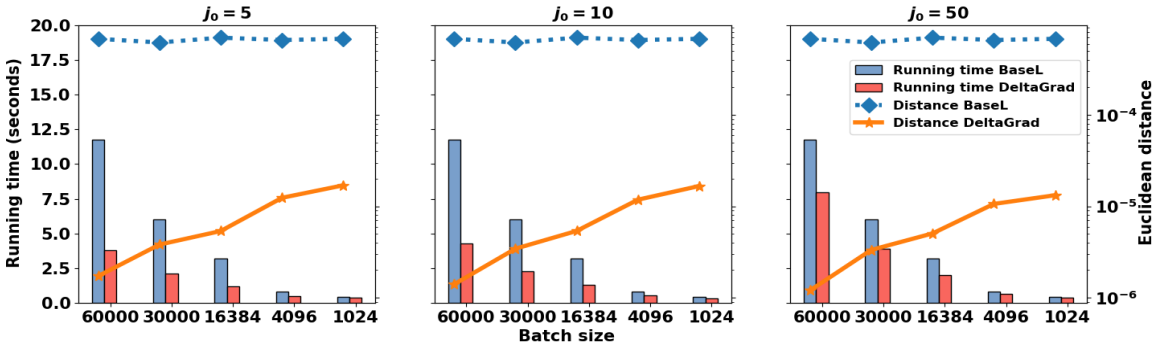


Figure 4.10 – Running time and distance comparison with varying mini-batch size under fixed $T_0 = 5$ and varying j_0 ($j_0 = 5$ VS $j_0 = 10$ VS $j_0 = 50$)

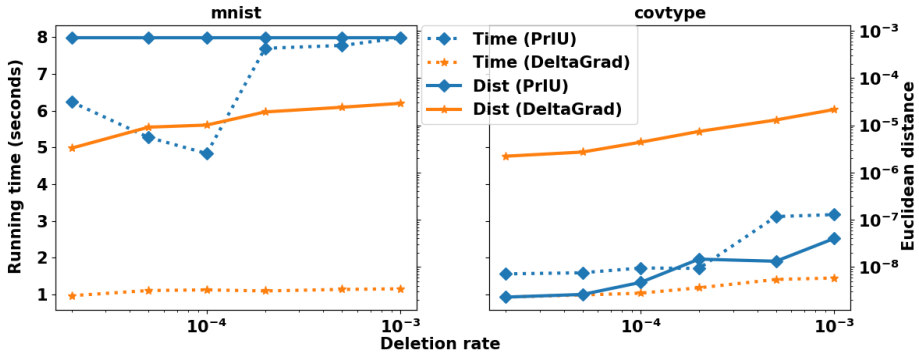


Figure 4.11 – Comparison of DeltaGrad and PrIU

proper hyper-parameters when DeltaGrad is used.

Influence of the mini-batch size. It is clear from Figure 4.9-4.10 that with larger mini-batch sizes, DeltaGrad can gain more speed with longer running time for both BaseL and DeltaGrad. As discussed before, to compute the gradients, other GPU-related overhead (the overhead to copy data from CPU DRAM to GPU DRAM, the time to launch the kernel on GPU) cannot be ignored. This can become more significant when compared against the smaller computational overhead for smaller mini-batch data. Also notice that, when $T_0 = 5$, with increasing B, the difference between $\mathbf{w}^U$ and $\mathbf{w}^I$ becomes smaller and smaller, which matches my conclusion in Theorem 11, i.e. with larger B, the difference $o(\frac{r}{n} + \frac{1}{B^{\frac{1}{4}}})$ is smaller.

Influence of T_0. By comparing the three sub-figures in Figure 4.9, the running time

slightly (rather than significantly) decreases with increasing T_0 for the same mini-batch size. This is explained by the earlier analysis in this section on the non-ideal performance for GPU computation over small matrices. Interestingly, when $T_0 = 10$ or $T_0 = 20$, $\|\mathbf{w}^{I,S} - \mathbf{w}^{U,S}\|$ does not decrease with larger mini-batch sizes. So to make the bound $o((\frac{r}{n} + \frac{1}{B^{\frac{1}{4}}}))$ hold, proper choice of T_0 is important. For example, $T_0 = 5$ is a good choice for MNIST dataset. This can achieve speed-ups comparable to larger T_0 without sacrificing the closeness between $\mathbf{w}^{I,S}$ and $\mathbf{w}^{U,S}$.

Influence of j_0. By comparing the three sub-figures in Figure 4.10, with increasing j_0, long "burn-in" iterations are expected, thus incurring more running time. This, however, does not significantly reduce the distance between $\mathbf{w}^{I,S}$ and $\mathbf{w}^{U,S}$. It indicates that I can select smaller j_0, e.g. 5 or 10 for more speed-up.

Discussions on tuning the hyper-parameters for DeltaGrad. Through my extensive experiments, I found that for regularized logistic regression, setting T_0 as 5 and j_0 as $5 - 20$ would lead to some of the most favorable trade-offs between running time and the error $\|\mathbf{w}^{U,S} - \mathbf{w}^{I,S}\|$. But in terms of more complicated models, e.g. 2-layer DNN, higher j_0 (even half of the total iteration number) and smaller T_0 (2 or 1) are necessary. Similar experiments were also conducted on adding training samples, in which similar trends were observed.

4.3.6.4 Comparison against the state-of-the-art work

In this subsection, I compared DeltaGrad (with $T_0 = 5$ and $j_0 = 10$) against PrIU by running regularized logistic regression over MNIST and covtype with the same mini-batch size (16384), the same learning rate and regularization rate, but with varying deletion rates.

The running time and the distance term $\|\mathbf{w}^U - \mathbf{w}^I\|$ of both PrIU and DeltaGrad with varying deletion rate are presented in Figure 4.11. First, it shows that DeltaGrad is always faster than PrIU, with more significant speed-ups on MNIST. The reason is that the time complexity of PrIU is $O(rp)$ for each iteration where p represents the total number of model parameters while r represents the reduced dimension after Singular Value Decomposition

is conducted over some $p \times p$ matrix. This is a large integer for large sparse matrices, e.g. MNIST.

As a result, $O(rp)$ is larger than the time complexity of DeltaGrad. Also, the memory usage of PrIU and DeltaGrad is shown in Table 4.10. PrIU needs much more DRAM (even 10x in MNIST) than DeltaGrad. The reason is that to prepare for the model update phase, PrIU needs to collect more information during the training phase over the full dataset. This is needed in the model update phase and is quadratic in the number of the model parameters p. As discussed in Section 4.2, PrIU cannot provide good performance over sparse datasets in terms of running time, error term $\mathbf{w}^U - \mathbf{w}^I$ and memory usage. In contrast, both the time and space overhead of DeltaGrad are smaller, which thus indicates the potential of its usage in the realistic, large-scale scenarios.

4.4 Acknowledgement

For this part of the dissertation, I collaborated closely with Dr. Val Tannen, who helped with the development and analysis of PrIU, and Dr. Edgar Dobrian, who helped with the formalization and analysis of DeltaGrad.

Algorithm 12: DeltaGrad (general models)

Input : The full training set $(\mathbf{X}, \mathbf{Y})$, model parameters cached during the training phase for the full training samples $\{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_t\}$ and corresponding gradients $\{\nabla F(\mathbf{w}_0), \nabla F(\mathbf{w}_1), \ldots, \nabla F(\mathbf{w}_t)\}$, the removed training sample or the added training sample R, period T_0, total iteration number T, history size m_0, warmup iteration number j_0, learning rate η

Output: Updated model parameter $\mathbf{w}_t^I$

1 Initialize $\mathbf{w}_0^I \leftarrow \mathbf{w}_0$
2 Initialize an array $\Delta G = []$
3 Initialize an array $\Delta W = []$
4 Initialize $last\_t = j_0$
5 $is\_explicit = False$
6 **for** $t = 0; t < T; t++$ **do**
7 **if** $(t - last_t) \mod T_0 == 0$ *or* $t \leq j_0$ **then**
8 $is\_explicit = True$
9 **else**
10 **end**
11 **if** $is\_explicit == True$ *or* $t \leq j_0$ **then**
12 $last\_t = t$
13 compute $\nabla F\left(\mathbf{w}_t^I\right)$ exactly
14 compute $\nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$ based on the cached gradient $\nabla F\left(\mathbf{w}_t\right)$
 `/* check local convexity */`
15 **if** $< \nabla F\left(\boldsymbol{w}_t^I\right) - \nabla F\left(\boldsymbol{w}_t\right), \boldsymbol{w}_t^I - \boldsymbol{w}_t >\leq 0$ **then**
16 compute $\mathbf{w}_{t+1}^I$ by using exact GD update (equation (4.45))
17 **continue**
18 **end**
19 set $\Delta G[k] = \nabla F\left(\mathbf{w}_t^I\right) - \nabla F\left(\mathbf{w}_t\right)$
20 set $\Delta W[k] = \mathbf{w}_t^I - \mathbf{w}_t$, based on the cached parameters $\mathbf{w}_t$
21 $k \leftarrow k + 1$
22 compute $\mathbf{w}_{t+1}^I$ by using exact GD update (equation (4.45))
23 **else**
24 Pass $\Delta W[-m:]$, $\Delta G[-m:]$, the last m_0 elements in ΔW and ΔG, which are from the $j_1^{th}, j_2^{th}, \ldots, j_m^{th}$ iterations where $j_1 < j_2 < \cdots < j_m$ depend on t, $\mathbf{v} = \mathbf{w}_t^I - \mathbf{w}_t$, and the history size m_0, to the L-BFGFS Algorithm (See Supplement) to get the approximation of $\mathbf{H}(\mathbf{w}_t)\mathbf{v}$, i.e., $\mathbf{B}_{j_m}\mathbf{v}$
 `/* check local smoothness */`
25 **if** $\|\boldsymbol{B}_{j_m}\boldsymbol{v}\| \geq \|\boldsymbol{v}\|$ **then**
26 **go to** line 12
27 **end**
28 Approximate $\nabla F\left(\mathbf{w}_t^I\right) = \nabla F\left(\mathbf{w}_t\right) + \mathbf{B}_{j_m}\left(\mathbf{w}_t^I - \mathbf{w}_t\right)$
29 Compute $\mathbf{w}_{t+1}^I$ by using the "leave-r-out" gradient formula, based on the approximated $\nabla F(\mathbf{w}_t^I)$
30 **end**
31 **end**
32 **return** $\mathbf{w}_t^I$

Table 4.8 – Prediction accuracy of BaseL and DeltaGrad with batch addition/deletion. MNIST^n refers to MNIST with a neural net.

Dataset		BaseL(%)	DeltaGrad(%)
Add (0.005%)	MNIST	87.530 ± 0.0025	87.530 ± 0.0025
	MNIST^n	92.340 ± 0.002	92.340 ± 0.002
	covtype	62.991 ± 0.0027	62.991 ± 0.0027
	HIGGS	55.372 ± 0.0002	55.372 ± 0.0002
	RCV1	92.222 ± 0.00004	92.222 ± 0.00004
Add (1%)	MNIST	87.540 ± 0.0011	87.542 ± 0.0011
	MNIST^n	92.397 ± 0.001	92.397 ± 0.001
	covtype	63.022 ± 0.0008	63.022 ± 0.0008
	HIGGS	55.381 ± 0.0007	55.380 ± 0.0007
	RCV1	92.233 ± 0.00010	92.233 ± 0.00010
Delete (0.005%)	MNIST	86.272 ± 0.0035	86.272 ± 0.0035
	MNIST^n	92.203 ± 0.004	92.203 ± 0.004
	covtype	62.966 ± 0.0017	62.966 ± 0.0017
	HIGGS	52.950 ± 0.0001	52.950 ± 0.0001
	RCV1	92.241 ± 0.00004	92.241 ± 0.00004
Delete (1%)	MNIST	86.082 ± 0.0046	86.074 ± 0.0048
	MNIST^n	92.373 ± 0.003	92.370 ± 0.003
	covtype	62.943 ± 0.0007	62.943 ± 0.0007
	HIGGS	52.975 ± 0.0002	52.975 ± 0.0002
	RCV1	92.203 ± 0.00007	92.203 ± 0.00007

Table 4.9 – Distance and prediction performance of BaseL and DeltaGrad in online deletion/addition

Dataset	Distance		Prediction accuracy (%)	
	$\|\mathbf{w}^{U*} - \mathbf{w}^*\|$	$\|\mathbf{w}^{I*} - \mathbf{w}^{U*}\|$	BaseL	DeltaGrad
MNIST (Addition)	5.7×10^{-3}	2×10^{-4}	87.548 ± 0.0002	87.548 ± 0.0002
MNIST (Deletion)	5.0×10^{-3}	1.4×10^{-4}	87.465 ± 0.002	87.465 ± 0.002
covtype (Addition)	8.0×10^{-3}	2.0×10^{-5}	63.054 ± 0.0007	63.054 ± 0.0007
covtype (Deletion)	7.0×10^{-3}	2.0×10^{-5}	62.836 ± 0.0002	62.836 ± 0.0002
HIGGS (Addition)	2.1×10^{-5}	1.4×10^{-6}	55.303 ± 0.0003	55.303 ± 0.0003
HIGGS (Deletion)	2.5×10^{-5}	1.7×10^{-6}	55.333 ± 0.0008	55.333 ± 0.0008
RCV1 (Addition)	0.0122	3.6×10^{-6}	92.255 ± 0.0003	92.255 ± 0.0003
RCV1 (Deletion)	0.0119	3.5×10^{-6}	92.229 ± 0.0006	92.229 ± 0.0006

Table 4.10 – Memory usage of DeltaGrad and PrIU(GB)

Deletion rate	MNIST		covtype	
	PrIU	DeltaGrad	PrIU	DeltaGrad
2×10^{-5}	26.61	2.74	9.30	2.56
5×10^{-5}	27.02	2.74	9.30	2.56
1×10^{-4}	27.13	2.74	9.30	2.55
2×10^{-4}	27.75	2.74	9.31	2.56
5×10^{-4}	29.10	2.74	10.67	2.56
1×10^{-3}	29.10	2.74	10.67	2.56

CHAPTER 5: Cleaning probabilistic labels with CHEF

In Chapter 4, I have discussed how to incrementally update ML models with PrIU and DeltaGrad. One use of those two methods is to facilitate the efficient evaluation of the training sample importance. In this chapter, I present how this goal is achieved in one application, i.e. in the application of cleaning probabilistic labels. As indicated in Chapter 1, the label cleaning pipeline is iterative, consisting of three phases, i.e. the *sample selector phase*, the *annotation phase* and the *model constructor phase*, which are described in detail next.

Sample selector phase. Finding the most influential training samples can be done with several different influence measures, e.g., the influence function [44], the Data Shapley values [113], the noisy label detection algorithms [46, 111], the active learning technique [45] or using a bi-level optimization solution [115]. Unfortunately, these do not work well for cleaning weak labels. I therefore develop a variant of the influence function called Infl which can simultaneously detect the most influential samples and suggest cleaned labels. One key technical challenge in the efficient implementation of Infl concerns the explicit evaluation of gradients on *every* training sample. **I address this challenge by developing Increm-INFL, which removes uninfluential training samples early and can thus *incrementally* recommend the most influential training samples to human annotators.**

Human annotation phase After influential samples are selected, the next step is for human annotators to clean the labels of those samples. Recall that *multiple* human annotators may be used to independently label each training sample, and inconsistencies between the labels are resolved, e.g., by majority vote [41]. **To reduce the cost of the human**

annotation phase, I consider the suggested clean labels from the *sample selector phase* as one alternative labeler, which can be combined with results provided by the human annotators to reduce annotation cost.

Model constructor phase. The previously described provenance-based algorithm DeltaGrad [95] can be used to incrementally update model parameters after the deletion or addition of a small subset of training samples. Since the result of the human annotation phase can be regarded as the deletion of top-B samples with probabilistic labels, and insertion of those same samples with cleaned labels, DeltaGrad can be adapted for this setting. This algorithm is called DeltaGrad-L. **To accelerate the model constructor phase, rather than retraining from scratch after cleaning the labels of a small set of training samples, I *incrementally* update the model using DeltaGrad-L.**

Redesign of the cleaning pipeline The final contribution of CHEF, which is enabled by the reduced cost of the sample selection, human annotation, and model construction phases, is a re-design of the pipeline in Figure 1.1 (see the loop (2)). Rather than providing all top-B influential training samples (and suggesting how to fix the label uncertainty) at once, the sample selector gives the human annotator the next top-b influential training samples, where b is smaller than B and is specified by the user. The model is then *refreshed* using the cleaned labels, and the next top-b samples to be given to the human annotator are calculated. This continues until the initial budget B has been exhausted or the expected prediction performance is reached (thus terminating early). **This can not only improve the overall model performance, but also lead to early termination, thus further saving the cost of human annotation.** Note that to enable incremental computation by Increm-INFL and DeltaGrad-L, some "provenance" information is necessary, and can be pre-computed offline in an *Initialization step* prior to the start of loop (2).

I demonstrate the effectiveness of CHEF using several crowd-sourced datasets as well as real medical image datasets. Our experiments show that CHEF achieves up to 54.7x speed-up in the sample selector phase, and up to 7.5x speed-up in the model constructor

phase. Furthermore, by using Infl and smaller batch sizes b, the overall model quality can be improved.

The rest of this chapter is organized as follows. Preliminary notation, definitions and assumptions are given in Section 5.1, followed by our algorithms, Infl, Increm-INFL and DeltaGrad-L in Section 5.2. Experimental results are discussed in Section 5.3.

5.1 Preliminaries

In this section, I introduce essential notation and assumptions, and then describe the influence function and DeltaGrad.

5.1.1 Notation

A C-class classification task is a classification task in which the number of classes is C. Suppose that the goal is to construct a machine learning model on a training set, $\mathcal{Z} = \mathcal{Z}_d \bigcup \mathcal{Z}_p$, in which $\mathcal{Z}_d = \{\mathbf{z}_i\}_{i=1}^{N_d} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_d}$ and $\mathcal{Z}_p = \{\tilde{\mathbf{z}}_i\}_{i=1}^{N_p} = \{(\tilde{\mathbf{x}}_i, \tilde{y}_i)\}_{i=1}^{N_p}$, denoting a set of N_d training samples with deterministic labels and N_p training samples with probabilistic labels, respectively. A *probabilistic label*, $\tilde{y}_i$, is represented by a probabilistic vector of length C, in which the value in the c_{th} entry ($c = 0, 1, \ldots, C-1$) denotes the probability that $\tilde{\mathbf{z}}_i$ belongs to the class c. The performance of the model constructed on $\mathcal{Z}$ is then validated on a validation dataset $\mathcal{Z}_{\text{val}}$ and tested on a test dataset $\mathcal{Z}_{\text{test}}$. Note that the size of $\mathcal{Z}_{\text{val}}$ and $\mathcal{Z}_{\text{test}}$ are typically small, consisting of samples with ground-truth labels or deterministic labels verified by the human annotators. Due to the possibly negative effect brought by the uncleaned training samples with probabilistic labels, it is reasonable to regularize those samples in the following objective function (e.g. see [149]):

$$F(\mathbf{w}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} F(\mathbf{w}, \mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma F(\mathbf{w}, \tilde{\mathbf{z}}_i)] \tag{5.1}$$

In the formula above, I use $\mathbf{w}$ to represent the model parameter, $F(\mathbf{w}, \mathbf{z})$ to denote the loss incurred on a sample $\mathbf{z}$ with the model parameter $\mathbf{w}$ and γ ($0 < \gamma < 1$, specified by

users) to denote the weight on the uncleaned training samples. Furthermore, the first order gradient of this loss can be denoted by $\nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z})$, and the second order gradient (i.e. the Hessian matrix) by $\mathbf{H}(\mathbf{w}, \mathbf{z})$. I further use $\nabla_{\mathbf{w}} F(\mathbf{w})$ and $\mathbf{H}(\mathbf{w})$ to denote the first order gradient and the Hessian matrix averaged over all weighted training samples.

To optimize Equation (5.1), stochastic Gradient Descent (SGD) can be applied. At each SGD iteration t, one essential step is to evaluate the first-order gradients of a randomly sampled mini-batch of training samples, $\mathcal{B}_t$ (I denote the size of $\mathcal{B}_t$ as $|\mathcal{B}_t|$), i.e.:

$$\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{B}_t) = \frac{1}{|\mathcal{B}_t|} \sum\nolimits_{\mathbf{z} \in \mathcal{B}_t} \gamma_{\mathbf{z}} \nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}),$$

in which γ_z is 1 if $z \in \mathcal{Z}_d$ and γ otherwise.

Since the loop (2) in Figure 1.1 may be repeated for multiple rounds, I use $\mathcal{Z}^{(k)}$ to denote the updated training dataset after k rounds and $\mathbf{w}^{(k)}$ to represent the model constructed on $\mathcal{Z}^{(k)}$.

5.1.2 Assumptions

I make two assumptions: the *strong convexity assumption*, and the *small cleaning budget assumption*.

Strong convexity assumption Following [95], I focus on model classes satisfying $\mu-$*strong convexity*, meaning that the minimal eigenvalue of each Hessian matrix $\mathbf{H}(\mathbf{w}, \mathbf{z})$ is always greater than a non-negative constant μ for arbitrary $\mathbf{w}$ and $\mathbf{z}$. One typical model satisfying this property is logistic regression with L2 norm regularization.

Small cleaning budget assumption Since manually cleaning labels is time-consuming and expensive, I assume that *the cleaning budget B is far smaller than the size of training set, $\mathcal{Z}$.*

5.1.3 Influence function

The influence function method [44] is originally proposed to estimate how the prediction performance on one test sample $\mathbf{z}_{\text{test}}$ is varied if I delete one training sample $\mathbf{z}$, or add an

infinitely small perturbation on the feature of $\mathbf{z}$. This is formulated as follows:

$$\mathcal{I}_{\text{del}}(\mathbf{z}) = -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}_{\text{test}})^{\top} \mathbf{H}^{-1}(\mathbf{w}) \nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z})$$
$$\mathcal{I}_{\text{pert}}(\mathbf{z}) = -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}_{\text{test}})^{\top} \mathbf{H}^{-1}(\mathbf{w}) \nabla_{\mathbf{x}} \nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}).$$

I can then leverage $\mathcal{I}_{\text{del}}(\mathbf{z})$ and $\mathcal{I}_{\text{pert}}(\mathbf{z})\delta$ to approximate the additional errors incurred on the test sample $\mathbf{z}_{\text{test}}$ after deleting the training sample $\mathbf{z}$, or perturbing the feature of $\mathbf{z}$ by δ.

As [44] indicates, by evaluating the training sample influence with the above influence function, the "harmful" training samples on the model prediction (i.e. the one with negative influence) can be distinguished from the "helpful" ones (i.e. the one with positive influence). I can then prioritize the most "harmful" training samples with probabilistic labels for cleaning. In practice, due to the invisibility of the test samples in most cases, the validation set is used instead, leading to the following modified influence functions:

$$\mathcal{I}_{\text{del}}(\mathbf{z}) = -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^{\top} \mathbf{H}^{-1}(\mathbf{w}) \nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}) \tag{5.2}$$
$$\mathcal{I}_{\text{pert}}(\mathbf{z}) = -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^{\top} \mathbf{H}^{-1}(\mathbf{w}) \nabla_{\mathbf{x}} \nabla_{\mathbf{w}} F(\mathbf{w}, \mathbf{z}) \tag{5.3}$$

The two formulas above also follow the modified influence function in [115] which uses a set of trusted validation samples instead of test samples to estimate the influence of each training sample.

5.2 Methodology

In this section, I describe the system design in detail for the sample selector phase (Section 5.2.1), the model constructor phase (Section 5.2.2) and the human annotation phase (Section 5.2.3).

5.2.1 The sample selector phase

Sample selection accomplishes two things: 1) it calculates the training sample influence using Infl in order to prioritize the most influential uncleaned training samples for cleaning,

and simultaneously suggests possibly cleaned labels for them (see Section 5.2.1.1); and 2) it filters out uninfluential training samples early using Increm-INFL at each round of loop (2) (see Section 5.2.1.3).

5.2.1.1 Infl

The goal of Infl is to calculate the influence of an uncleaned training sample, $\tilde{\mathbf{z}}$, by estimating how much additional error will be incurred on the validation set $\mathcal{Z}_{\text{val}}$ if 1) the probabilistic label of $\tilde{\mathbf{z}}$ is updated to some deterministic label; and 2) $\tilde{\mathbf{z}}$ is up-weighted to 1 after it is cleaned, which is similar to (but fully not covered by) the intuition of the influence function method [44]. To capture this intuition, I propose the following modified influence function (its derivation is postponed until Section 5.2.1.2):

$$\begin{aligned}&\mathcal{I}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) \approx N \cdot (F(\mathbf{w}^U, \mathcal{Z}_{\text{val}}) - F(\mathbf{w}, \mathcal{Z}_{\text{val}}))\\ &= -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^\top \mathbf{H}^{-1}(\mathbf{w})[\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})\delta_y + (1-\gamma)\nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})],\end{aligned} \tag{5.4}$$

in which δ_y denotes the difference between the original probabilistic label of $\tilde{\mathbf{z}}$ and one deterministic label (ranging from 0 to $C-1$) and $\mathbf{w}^U$ denotes the updated model parameters after the label is cleaned and $\tilde{\mathbf{z}}$ is up-weighted. To calculate δ_y, the deterministic label is first converted to its one-hot representation, i.e. a vector of length C taking 1 in the c_{th} entry ($c = 0, 1, \ldots, C-1$) for the label c and taking 0 in all other entries (recall that C represents the number of classes).

To recommend the most influential uncleaned training samples to the human annotators and suggest possibly cleaned labels, I 1) explicitly evaluate Equation (5.4) for each uncleaned training sample for *all possible deterministic labels*, 2) prioritize the most "harmful" training samples for cleaning, i.e. the ones with the smallest negative influence values after their labels are updated to *some* deterministic labels, and 3) suggest those deterministic labels as the potentially cleaned labels for the human annotators.

Comparison to [115] As discussed earlier, DUTI [115] can also recommend the most influential training samples for cleaning and suggest possibly cleaned labels, which is ac-

complished through solving a bi-level optimization problem. However, solving this problem is computationally challenging, and therefore this method cannot be used in real-time over multiple rounds (i.e. in loop (2)).

The authors of [115] also modified the influence function to reflect the perturbations of the training labels as follows:

$$\mathcal{I}_{\text{pert}}(\tilde{\mathbf{z}}) = -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^{\top} \mathbf{H}^{-1}(\mathbf{w}) \nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}}), \tag{5.5}$$

and compared it against DUTI. Equation (5.5) is equivalent to removing δ_y (which quantifies the effect of label changes) and $(1-\gamma)\nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})$ from Equation (5.4). As will be shown in Section 5.3, ignoring δ_y in Equation (5.5) can lead to worse performance than Infl even when all the training samples are equally weighted.

Computing $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})$ At first glance, it seems that the term $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})$ cannot be calculated using auto-differentiation packages such as Pytorch, since it involves the partial derivative with respect to the label of $\tilde{\mathbf{z}}$. However, I notice that this partial derivative can be explicitly calculated when the loss function $F(\mathbf{w}, \tilde{\mathbf{z}})$ is the cross-entropy function, which is the most widely used objective function in the classification task. Specifically, the instantiation of the loss function $F(\mathbf{w}, \tilde{\mathbf{z}})$ into the cross-entropy function can be expressed as:

$$F(\mathbf{w}, \tilde{\mathbf{z}}) = -\sum\nolimits_{k=1}^{C} \tilde{y}^{(k)} \log(p^{(k)}(\mathbf{w}, \tilde{\mathbf{x}})), \tag{5.6}$$

In this formula above, $\tilde{y} = [\tilde{y}^{(1)}, \tilde{y}^{(2)}, \ldots, \tilde{y}^{(C)}]$ is the label of an input sample $\tilde{\mathbf{z}} = (\tilde{\mathbf{x}}, \tilde{y})$ and $[p^{(1)}(\mathbf{w}, \tilde{\mathbf{x}}), p^{(2)}(\mathbf{w}, \tilde{\mathbf{x}}), \ldots, p^{(C)}(\mathbf{w}, \tilde{\mathbf{x}})]$ represents the model output given this input sample, which is a probabilistic vector of length C depending on the model parameter $\mathbf{w}$ and the input feature $\tilde{\mathbf{x}}$. Then I can observe that Equation (5.6) is a linear function of the label $\tilde{y}$. Hence, $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}})$ can be explicitly evaluated as:

$$\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}, \tilde{\mathbf{z}}) = [-\nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}, \tilde{\mathbf{x}})), \ldots, -\nabla_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}, \tilde{\mathbf{x}}))] \tag{5.7}$$

As a result, each $-\nabla_{\mathbf{w}} \log(p^{(c)}(\mathbf{w}, \tilde{\mathbf{x}})), c = 0, 1, \ldots, C-1$ can be calculated with the auto-differentiation package.

Computing $\mathbf{H}^{-1}(\mathbf{w})$ Recall that $\mathbf{H}(\mathbf{w})$ denotes the Hessian matrix averaged on all training samples. Rather than explicitly calculating its inverse, by following [44], I leverage the conjugate gradient method [150] to approximately compute the Matrix-vector product $\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^{\top} \mathbf{H}^{-1}(\mathbf{w})$ in Equation (5.4).

5.2.1.2 Derivation of Equation (5.4)

According to [44], to analyze the influence of the label changes on one training sample $\tilde{\mathbf{z}}$ as well as re-weighting this sample, I need to consider the following objective function:

$$F_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}(\mathbf{w}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} F(\mathbf{w}, \mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma F(\mathbf{w}, \tilde{\mathbf{z}}_i)] + \epsilon_1 F(\mathbf{w}, \tilde{\mathbf{z}}(\delta_y)) - \epsilon_2 F(\mathbf{w}, \tilde{\mathbf{z}}) \quad (5.8)$$

in which $\tilde{\mathbf{z}} = (\tilde{\mathbf{x}}, \tilde{y}) \in \mathcal{Z}_p = \{\tilde{\mathbf{z}}_i\}_{i=1}^{N_p}$, $\tilde{\mathbf{z}}(\delta_y) = (\tilde{\mathbf{x}}, \tilde{y} + \delta_y)$, representing the $\tilde{\mathbf{z}}$ with the cleaned label $\tilde{y} + \delta_y$, and ϵ_1 and ϵ_2 are two small weights. I can adjust the values of ϵ_1 and ϵ_2 to obtain a new objective function such that the effect of the $\tilde{\mathbf{z}}$ is cancelled out and its cleaned version is up-weighted. To achieve this, I can set $\epsilon_1 = \frac{1}{N}$ and $\epsilon_2 = \frac{\gamma}{N}$.

Then when Equation (5.8) is minimized, its gradient should be zero. Then by denoting its minimizer as $\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}$, the following equation holds:

$$\begin{aligned}\nabla_{\mathbf{w}} F_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}(\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}, \mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}, \tilde{\mathbf{z}}_i)] \\ + \epsilon_1 \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}, \tilde{\mathbf{z}}(\delta_y)) - \epsilon_2 \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}, \tilde{\mathbf{z}}) = 0\end{aligned}$$

I also denote the minimizer of $\text{argmin}_{\mathbf{w}} F_{0,0,\tilde{\mathbf{z}}}(\mathbf{w})$ as $\hat{\mathbf{w}}$, which is also the minimizer of Equation (5.1) and is derived before any training sample is cleaned. Due to the closeness of $\hat{\mathbf{w}}_{\epsilon_1, \epsilon_2, \tilde{\mathbf{z}}}$[1] and $\hat{\mathbf{w}}$ as both ϵ_1 and ϵ_2 are near-zero values, I can then apply Taylor expansion

[1] this is one implicit assumption of the influence function method

on $\nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}, \epsilon_1, \epsilon_2\right)$, i.e.:

$$\begin{aligned}
0 &= \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}, \epsilon_1, \epsilon_2\right) \approx \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \epsilon_1, \epsilon_2\right) + \mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)\left(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}} - \hat{\mathbf{w}}\right) \\
&= \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \mathbf{z}_i\right) + \sum\nolimits_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}_i\right)] \\
&+ \epsilon_1 \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}(\delta_y)\right) - \epsilon_2 \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) + \mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)\left(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}} - \hat{\mathbf{w}}\right),
\end{aligned} \tag{5.9}$$

in which $\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}(*)$ denotes the Hessian matrix of $F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}(\mathbf{w})$. Then by using the fact that $\frac{1}{N}[\sum_{i=1}^{N_d} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \mathbf{z}_i\right) + \sum_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}_i\right)] = 0$ (since $\hat{\mathbf{w}}$ is the minimizer of $F_{0,0,\tilde{\mathbf{z}}}(\mathbf{w})$) and $\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right) \approx \mathbf{H}_{0,0,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right) = \mathbf{H}\left(\hat{\mathbf{w}}\right)$ (since ϵ_1 and ϵ_2 are near zero, recall that $\mathbf{H}(*)$ is the Hessian matrix of Equation (5.1)), the formula above is derived as:

$$\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}} - \hat{\mathbf{w}} = -\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\epsilon_1 \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}(\delta_y)\right) - \epsilon_2 \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right] \tag{5.10}$$

Recall that $\epsilon_1 = \frac{1}{N}$ and $\epsilon_2 = \frac{\gamma}{N}$ for the purpose of cleaning the labels of $\tilde{\mathbf{z}}$ and re-weighting it afterwards. Then the formula above is further reformulated as:

$$\hat{\mathbf{w}}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}} - \hat{\mathbf{w}} = -\mathbf{H}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}(\delta_y)\right) - \frac{\gamma}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right]$$

By further reorganizing the formula above and utilize the Cauchy mean value theorem, the following formula could be derived:

$$\begin{aligned}
&\hat{\mathbf{w}}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}} - \hat{\mathbf{w}} = -\mathbf{H}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}(\delta_y)\right) - \frac{\gamma}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right] \\
&= -\mathbf{H}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}(\delta_y)\right) - \frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) + \frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) - \frac{\gamma}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right] \\
&= -\mathbf{H}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\frac{1}{N} \nabla_{\mathbf{w}} \nabla_y F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) \delta_y + \frac{1}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) - \frac{\gamma}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right] \\
&= -\mathbf{H}_{\frac{1}{N},\frac{\gamma}{N},\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\frac{1}{N} \nabla_{\mathbf{w}} \nabla_y F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right) \delta_y + \frac{1-\gamma}{N} \nabla_{\mathbf{w}} F\left(\hat{\mathbf{w}}, \tilde{\mathbf{z}}\right)\right]
\end{aligned} \tag{5.11}$$

Recall that the influence function is to quantify how much the loss on the validation dataset varies after $\tilde{\mathbf{z}}$ is cleaned and re-weighted. Therefore, this version of the influence

function is obtained as follows:

$$\begin{aligned}
&\mathcal{I}_{\text{pert}}(\mathbf{z}, \delta_y, \gamma) = N \cdot (F(\hat{\mathbf{w}}_{\frac{1}{N}, \frac{\gamma}{N}, \tilde{\mathbf{z}}}, \mathcal{Z}_{\text{val}}) - F(\hat{\mathbf{w}}, \mathcal{Z}_{\text{val}})) \\
&\approx N \cdot (\nabla_{\mathbf{w}} F(\hat{\mathbf{w}}, \mathcal{Z}_{\text{val}})(\hat{\mathbf{w}}_{\frac{1}{N}, \frac{\gamma}{N}, \tilde{\mathbf{z}}} - \hat{\mathbf{w}})) \\
&= -\nabla_{\mathbf{w}} F(\hat{\mathbf{w}}, \mathcal{Z}_{\text{val}})^\top \mathbf{H}^{-1}(\hat{\mathbf{w}})[\nabla_y \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}, \mathbf{z}) \delta_y + (1-\gamma) \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}, \mathbf{z})],
\end{aligned}$$

5.2.1.3 Increm-INFL

The goal of using Infl is to quantify the influence of all uncleaned training samples and select the Top-b influential training samples for cleaning. But in loop (2), this search space could be reduced by employing Increm-INFL. Specifically, other than the initialization step, I can leverage *Increm-INFL* to prune away most of the uninfluential training samples early in following rounds, thus only evaluating the influence of a small set of candidate influential training samples in those rounds. Suppose this set of samples is denoted as $\mathcal{Z}_{inf}^{(k)}$ for the round k; the derivation of this set is outlined in Algorithm 14. As this algorithm indicates, the first step is to effectively estimate the maximal perturbations of Equation (5.4) at the k_{th} cleaning round for each uncleaned training sample $\tilde{\mathbf{z}}$ and each possible label change δ_y (see line 2), which are assumed to take $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ (see Theorem 13 for its definition) as the perturbation center. Then the first part of $\mathcal{Z}_{inf}^{(k)}$ consists of all the training samples which produce the Top-b smallest values of $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ with a given δ_y (see line 6). For those b smallest values, I also collect the maximal value of their upper bound, L. I then include in $\mathcal{Z}_{inf}^{(k)}$ all the remaining training samples whose lower bound, is smaller than L with certain δ_y (see line 4). This indicates the possibility of those samples becoming the Top-b influential samples.

To intuitively illustrate the above process to obtain $\mathcal{Z}_{inf}^{(k)}$, I provided an example in Figure 5.1. In this figure, I use $\mathbf{I}_1 \leq \mathbf{I}_2 \leq \mathbf{I}_3 \leq \ldots$ to denote the sorted list of $\{\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma) | \tilde{\mathbf{z}} = (\tilde{\mathbf{x}}, \tilde{y}) \in \mathcal{Z}_p, \delta_y = \tilde{y} - c, c \in \{0, 1, \ldots, C-1\}\}$ calculated among all the training samples and all possible label perturbations. As described in Section 5.2.1.3, the set of candidate influential training samples consists of two parts, one comprised of training samples

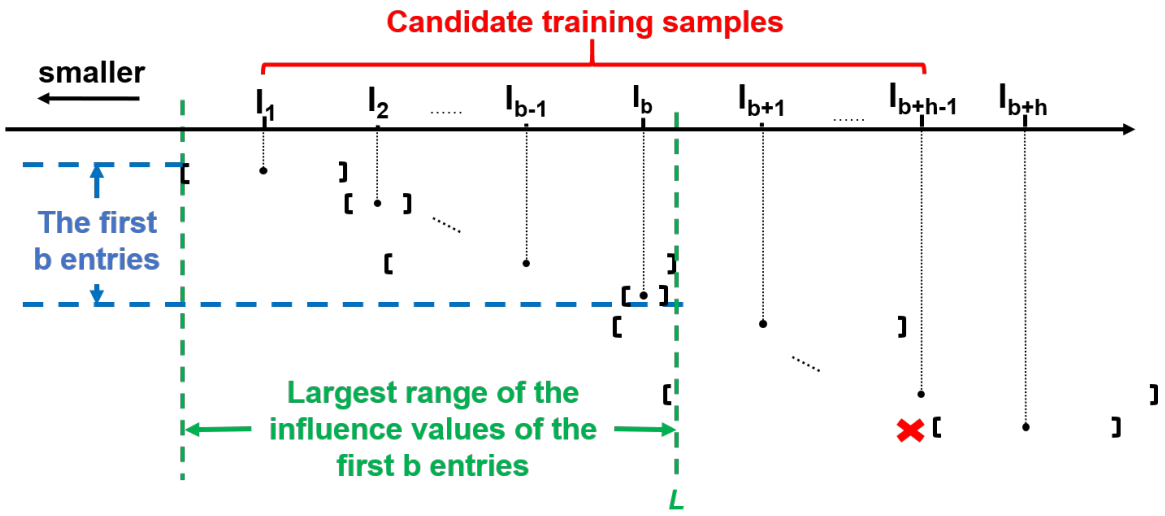


Figure 5.1 – Intuitive illustration of Increm-INFL

producing Top-b smallest values of $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$, i.e., the training samples generating the value $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \ldots, \mathbf{I}_b$ for $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$. The other part includes all the other training samples whose lower bound on $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ is smaller than the largest upper bound of the items, $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \ldots, \mathbf{I}_b$. For example, in Figure 5.1, the training samples corresponding to the value, $\mathbf{I}_{b+1}, \mathbf{I}_{b+2}, \mathbf{I}_{b+3}, \ldots, \mathbf{I}_{b+h-1}$ will become the candidate training samples while the sample producing value, $\mathbf{I}_{b+h}$, will not be counted as the candidate influential sample.

As described above, it is critical to estimate the maximal perturbation of Equation (5.4) for each uncleaned training sample, $\tilde{\mathbf{z}}$, and each label perturbation, δ_y, which requires the following theorem.

Theorem 13. *For a training sample $\tilde{\boldsymbol{z}} = (\tilde{\boldsymbol{x}}, \tilde{y})$ which has not been cleaned before the k_{th} round of loop (2), the following bounds hold for Equation (5.4) evaluated on the training sample $\tilde{\boldsymbol{z}}$ and a label perturbation δ_y:*

$$
\begin{aligned}
&| - \mathcal{I}^{(k)}_{pert}(\tilde{\boldsymbol{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\boldsymbol{z}}, \delta_y, \gamma) - \frac{1-\gamma}{2} e_1 \mu - \sum_{j=1}^{C} \delta_{y,j} e_1 \| \boldsymbol{H}^{(j)}(\boldsymbol{w}^{(k)}, \tilde{\boldsymbol{z}}) \| | \\
&\leq \sum_{j=1}^{C} |\delta_{y,j}| e_2 \| \boldsymbol{H}^{(j)}(\boldsymbol{w}^{(k)}, \tilde{\boldsymbol{z}}) \| + \frac{1-\gamma}{2} e_2 \mu
\end{aligned}
$$

in which, $\mathcal{I}_0(\tilde{\boldsymbol{z}}, \delta_y, \gamma) = \boldsymbol{v}^\top [\nabla_y \nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(0)}, \tilde{\boldsymbol{z}}) \delta_y + (1-\gamma) \nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(0)}, \tilde{\boldsymbol{z}})]$,

$\boldsymbol{v}^\top = -\nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(k)}, \mathcal{Z}_{val})^\top \boldsymbol{H}^{-1}(\boldsymbol{w}^{(k)})$, $\delta_y = [\delta_{y,1}, \delta_{y,2}, \ldots, \delta_{y,C}]$,
$\boldsymbol{H}^{(j)}(\boldsymbol{w}^{(k)}, \tilde{\boldsymbol{z}}) = \int_0^1 -\nabla^2_{\boldsymbol{w}} \log(p^{(j)}(\boldsymbol{w}^{(0)} + s(\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)}), \tilde{\boldsymbol{x}}))ds$,
$\mu = \| \int_0^1 \boldsymbol{H}(\boldsymbol{w}^{(0)} + s(\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)}), \tilde{\boldsymbol{z}})ds \|$, *and,* $e_1 = \boldsymbol{v}^\top(\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)})$, $e_2 = \|\boldsymbol{v}\| \|\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)}\|$.

Proof. Recall that $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma) = \mathbf{v}^\top \nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z}) \delta_y$, then the following equation holds:

$$
\begin{aligned}
&(-\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)) \\
&= \mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) \delta_y + (1-\lambda) \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})] \\
&- \mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z}) \delta_y + (1-\lambda) \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})] \\
&= \underbrace{\mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) \delta_y - \nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z}) \delta_y]}_{\text{Diff}_1} \\
&+ (1-\lambda) \underbrace{\mathbf{v}^\top [\nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) - \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})]}_{\text{Diff}_2}
\end{aligned}
\tag{5.12}
$$

Then by plugging the definition of $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})$ into the formula Diff_1 above, I can get:

$$
\begin{aligned}
\text{Diff}_1 &= \mathbf{v}^\top [-[\nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(0)}, \mathbf{x}))] \\
&, \ldots, -[\nabla_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}^{(0)}, \mathbf{x}))]] \delta_y
\end{aligned}
$$

Then by utilizing the Cauchy mean value theorem, the formula above can be rewritten as:

$$
\begin{aligned}
&\text{Diff}_1 = \mathbf{v}^\top [-[\nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(0)}, \mathbf{x}))] \\
&, \ldots, -[\nabla_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}^{(0)}, \mathbf{x}))]] \delta_y \\
&= \mathbf{v}^\top [\int_0^1 -\nabla^2_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}, \mathbf{x}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds (\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \\
&, \ldots, \int_0^1 -\nabla^2_{\mathbf{w}} \log(p^{(C)}(\mathbf{w}, \mathbf{x}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})] \delta_y \\
&= \mathbf{v}^\top [\mathbf{H}^{(1)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \ldots, \mathbf{H}^{(C)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})] \delta_y
\end{aligned}
$$

Then by using the definition of δ_y, i.e. $\delta_y = [\delta_{y,1}, \delta_{y,2}, \ldots, \delta_{y,C}]$, the formula above can be further derived as:

$$\text{Diff}_1 = \sum_{j=1}^{C} \delta_{y,j} \mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}). \tag{5.13}$$

Note that since each $\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z}), j = 0, 1, \ldots, C-1$ is a semi-positive definite matrix for strongly convex models, it can thus be decomposed with its eigenvalues and eigenvectors, i.e.:

$$\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z}) = \sum_{s=1}^{m} \sigma_s \mathbf{u}_s \mathbf{u}_s^\top$$

Therefore, for each summed term in Equation (5.13), it can be rewritten as below by using the formula above:

$$\begin{aligned} &\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) = \mathbf{v}^\top (\sum_{s=1}^{m} \sigma_s \mathbf{u}_s \mathbf{u}_s^\top)(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \\ &= \sum_{s=1}^{m} \sigma_s \mathbf{v}^\top \mathbf{u}_s \mathbf{u}_s^\top (\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \end{aligned} \tag{5.14}$$

Since $\mathbf{v}^\top \mathbf{u}_s$ and $\mathbf{u}_s^\top (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})$ are two scalars, they can be rewritten as $\mathbf{u}_s^\top \mathbf{v}$ and $(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s$ respectively. As a result, the formula above can be rewritten as:

$$\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) = \sum_{s=1}^{m} \sigma_s \mathbf{u}_s^\top \mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s, \tag{5.15}$$

Then for each summed term above, it is still a scalar. Therefore, I can also rewrite it as follows by introducing its transpose:

$$\begin{aligned} &\mathbf{u}_s^\top \mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s = \frac{1}{2} \left[(\mathbf{u}_s^\top \mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s + (\mathbf{u}_s^\top \mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s)^\top \right] \\ &= \frac{1}{2} \left[\mathbf{u}_s^\top \mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top \mathbf{u}_s + \mathbf{u}_s^\top (\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \mathbf{v}^\top \mathbf{u}_s \right] \\ &= \frac{1}{2} \mathbf{u}_s^\top [\mathbf{v} (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top + (\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \mathbf{v}^\top] \mathbf{u}_s \end{aligned} \tag{5.16}$$

Note that $[\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top+(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})\mathbf{v}^\top]$ is a symmetric matrix, which has orthogonal eigenvectors and thus can be decomposed with its eigenvectors as follows:

$$[\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top+(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})\mathbf{v}^\top]=\tilde{\mathbf{U}}\mathbf{A}\tilde{\mathbf{U}}^\top=\sum_{t=1}^{m}a_t\tilde{\mathbf{u}}_t\tilde{\mathbf{u}}_t^\top$$

in which $a_1 \geq a_2 \geq \cdots \geq a_m$ are the eigenvalues and each $\tilde{\mathbf{u}}_t$ is a mutually orthogonal eigenvector. The formula above is then plugged into Equation (5.16), which results in:

$$\begin{aligned}
&\mathbf{u}_s^\top\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top\mathbf{u}_s=\frac{1}{2}\left[\mathbf{u}_s^\top\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top\mathbf{u}_s+(\mathbf{u}_s^\top\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top\mathbf{u}_s)^\top\right]\\
&=\frac{1}{2}\left[\mathbf{u}_s^\top\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top\mathbf{u}_s+\mathbf{u}_s^\top(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})\mathbf{v}^\top\mathbf{u}_s\right]\\
&=\frac{1}{2}\mathbf{u}_s^\top[\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top+(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})\mathbf{v}^\top]\mathbf{u}_s\\
&=\frac{1}{2}\mathbf{u}_s^\top[\sum_{t=1}^{m}a_t\tilde{\mathbf{u}}_t\tilde{\mathbf{u}}_t^\top]\mathbf{u}_s
\end{aligned}$$

This formula is then plugged into Equation (5.15), leading to:

$$\begin{aligned}
&\mathbf{v}^\top\mathbf{H}^{(j)}(\mathbf{w}^{(k)},\mathbf{z})(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})=\sum_{s=1}^{m}\sigma_s\mathbf{u}_s^\top\mathbf{v}(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})^\top\mathbf{u}_s\\
&=\sum_{s=1}^{m}\sigma_s\left[\frac{1}{2}\mathbf{u}_s^\top[\sum_{t=1}^{m}\mathbf{a}_t\tilde{\mathbf{u}}_t\tilde{\mathbf{u}}_t^\top]\mathbf{u}_s\right]=\frac{1}{2}\sum_{s=1}^{m}\sum_{t=1}^{m}\sigma_s a_t\mathbf{u}_s^\top\tilde{\mathbf{u}}_t\tilde{\mathbf{u}}_t^\top\mathbf{u}_s\\
&=\frac{1}{2}\sum_{s=1}^{m}\sum_{t=1}^{m}\sigma_s a_t\tilde{\mathbf{u}}_t^\top\mathbf{u}_s\mathbf{u}_s^\top\tilde{\mathbf{u}}_t=\frac{1}{2}\sum_{t=1}^{m}a_t\tilde{\mathbf{u}}_t^\top\left[\sum_{s=1}^{m}\sigma_s\mathbf{u}_s\mathbf{u}_s^\top\right]\tilde{\mathbf{u}}_t
\end{aligned}\tag{5.17}$$

Recall that

$$\mathbf{H}^{(j)}(\mathbf{w}^{(k)},\mathbf{z})=\sum_{s=1}^{m}\sigma_s\mathbf{u}_s\mathbf{u}_s^\top$$

, which is a semi-definite positive matrix. As a result, the following inequality holds for arbitrary vector $\mathbf{u}$:

$$0\leq\mathbf{u}^\top\mathbf{H}^{(j)}(\mathbf{w}^{(k)},\mathbf{z})\mathbf{u}\leq\|\mathbf{H}^{(j)}(\mathbf{w}^{(k)},\mathbf{z})\|\mathbf{u}^\top\mathbf{u}=\|\mathbf{H}^{(j)}(\mathbf{w}^{(k)},\mathbf{z})\|\|\mathbf{u}\|^2$$

Therefore, Equation (5.17) can be bounded as:

$$\begin{aligned}\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) &\leq \frac{1}{2}\sum_{a_t \geq 0} a_t \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| \|\tilde{\mathbf{u}}_t\|^2 + \frac{1}{2}\sum_{a_t < 0} 0 \\ = \frac{1}{2}\sum_{a_t \geq 0} a_t \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| &= \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| \frac{1}{2}\sum_{a_t \geq 0} a_t\end{aligned} \tag{5.18}$$

and:

$$\begin{aligned}\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) &\geq \frac{1}{2}\sum_{a_t < 0} a_t \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| \|\tilde{\mathbf{u}}_t\|^2 + \frac{1}{2}\sum_{a_t \geq 0} 0 \\ = \frac{1}{2}\sum_{a_t < 0} a_t \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| &= \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| \frac{1}{2}\sum_{a_t < 0} a_t\end{aligned} \tag{5.19}$$

Note that the two non-zero eigenvalues of $\mathbf{v}(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})^\top + (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\mathbf{v}^\top$ are $\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \pm \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|$, which correspond to the eigenvectors $\|\mathbf{v}\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \pm \|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|\mathbf{v}$. For those two non-zero eigenvalues, $\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|$ is greater than 0 while $\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|$ is smaller than 0. Therefore, I can explicitly derive $\frac{1}{2}\sum_{a_t \geq 0} a_t$ and $\frac{1}{2}\sum_{a_t < 0} a_t$ as follows:

$$\begin{aligned}\frac{1}{2}\sum_{a_t \geq 0} a_t &= \frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|] \\ \frac{1}{2}\sum_{a_t < 0} a_t &= \frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]\end{aligned}$$

As a result, Equation (5.18) and Equation (5.19) can be further bounded as:

$$\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \leq \frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]\|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\|$$

and:

$$\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \geq \frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]\|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\|$$

Based on the results above, I can then derive the upper bound of Equation (5.13) as

follows:

$$\begin{aligned}
\text{Diff}_1 &= \sum_{j=1}^{C} \delta_{y,j} \mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \\
&\leq \sum_{\delta_{y,j} \geq 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \sum_{\delta_{y,j} < 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]]
\end{aligned} \tag{5.20}$$

Similarly, the lower bound of Equation (5.13) is derived as:

$$\begin{aligned}
\text{Diff}_1 &\geq \sum_{\delta_{y,j} < 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \sum_{\delta_{y,j} \geq 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]]
\end{aligned} \tag{5.21}$$

Then I move on to derive the bounds on Diff_2 in Equation (5.12). As the first step, I utilize the Cauchy mean value theorem on this term as follows:

$$\begin{aligned}
\text{Diff}_2 &= \mathbf{v}^\top [\nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) - \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})] \\
&= \mathbf{v}^\top [\int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds](\mathbf{w}^{(k)} - \mathbf{w}^{(0)})
\end{aligned}$$

which thus follows the same form as Equation (5.14). Therefore, by following the same derivation of the bounds on Equation (5.14), the formula above is bounded as:

$$\begin{aligned}
&\text{Diff}_2 = \mathbf{v}^\top [\nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) - \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})] \\
&\in \Big[\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|\mathbf{w}^{(k)} - \mathbf{w}^{(0)}\|]\| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds\|, \\
&\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|\mathbf{w}^{(k)} - \mathbf{w}^{(0)}\|]\| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds\|\Big]
\end{aligned} \tag{5.22}$$

As a consequence, by utilizing the results in Equation (5.20), (5.21) and (5.22), Equation

(5.12) is bounded as:

$$
\begin{aligned}
&\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma) \\
&\leq \sum_{\delta_{y,j} \geq 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\| \|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \sum_{\delta_{y,j} < 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\| \|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \frac{1-\gamma}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\| \|\mathbf{w}^{(k)} - \mathbf{w}^{(0)}\|] \| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds \|
\end{aligned}
$$

Then by denoting $e_1 = \mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})$ and $e_2 = \|\mathbf{v}\| \|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|$, the upper bound of $\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ can be denoted as:

$$
\begin{aligned}
&\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma) \\
&\leq \sum_{\delta_{y,j} \geq 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| (e_1 + e_2) + \sum_{\delta_{y,j} < 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| (e_1 - e_2) \\
&+ \frac{1-\gamma}{2}(e_1 + e_2) \| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds \| \\
&= \sum_{j=1}^{C} [\delta_{y,j} e_1 + |\delta_{y,j}| e_2] \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| + \frac{1-\gamma}{2}(e_1 + e_2) \| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds \|
\end{aligned}
$$

Similarly, I can derive the lower bound of Equation (5.12), i.e.:

$$
\begin{aligned}
&\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma) \\
&\geq \sum_{\delta_{y,j}<0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) + \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \sum_{\delta_{y,j}\geq 0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| [\frac{1}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})\|]] \\
&+ \frac{1-\gamma}{2}[\mathbf{v}^\top(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) - \|\mathbf{v}\|\|\mathbf{w}^{(k)} - \mathbf{w}^{(0)}\|]\| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds\| \\
&= \sum_{\delta_{y,j}<0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| (e_1 + e_2) + \sum_{\delta_{y,j}>0} \delta_{y,j} \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| (e_1 - e_2) \\
&+ \frac{1-\gamma}{2}(e_1 - e_2)\| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds\| \\
&= \sum_{j=1}^{C} [\delta_{y,j} e_1 - |\delta_{y,j}| e_2] \|\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})\| \\
&+ \frac{1-\gamma}{2}(e_1 - e_2)\| \int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \mathbf{z}) ds\|
\end{aligned}
$$

□

Lemma 4. *if the model is a binary logistic regression model (meaning that $C = 2$), then for a training sample $\tilde{\boldsymbol{z}} = (\tilde{\boldsymbol{x}}, \tilde{y})$ which has not been cleaned before the k^{th} round of loop (2), the following bounds hold for Equation* (5.4) *evaluated on the training sample $\tilde{\boldsymbol{z}}$ and a label perturbation δ_y:*

$$
| - \mathcal{I}^{(k)}_{pert}(\tilde{\boldsymbol{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\boldsymbol{z}}, \delta_y, \gamma) - \frac{1-\gamma}{2} e_1 \mu \| \leq \frac{1-\gamma}{2} e_2 \mu
$$

in which, $\mathcal{I}_0(\tilde{\boldsymbol{z}}, \delta_y, \gamma) = \boldsymbol{v}^\top [\nabla_y \nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(0)}, \tilde{\boldsymbol{z}}) \delta_y + (1-\gamma) \nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(0)}, \tilde{\boldsymbol{z}})]$,
$\boldsymbol{v}^\top = -\nabla_{\boldsymbol{w}} F(\boldsymbol{w}^{(k)}, \mathcal{Z}_{val})^\top \boldsymbol{H}^{-1}(\boldsymbol{w}^{(k)})$, $\delta_y = [\delta_{y,0}, \delta_{y,1}]$,
$\mu = \| \int_0^1 \boldsymbol{H}(\boldsymbol{w}^{(0)} + s(\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)}), \tilde{\boldsymbol{z}}) ds\|$, *and,* $e_1 = \boldsymbol{v}^\top(\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)})$, $e_2 = \|\boldsymbol{v}\| \|\boldsymbol{w}^{(k)} - \boldsymbol{w}^{(0)}\|$.

Proof. Similar to the proof of Theorem 13, I start by calculating the difference between

$-\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma)$ and $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$, i.e.:

$$
\begin{aligned}
&(-\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)) \\
&= \mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) \delta_y + (1-\lambda) \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})] - \\
&\mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z}) \delta_y + (1-\lambda) \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})] \\
&= \underbrace{\mathbf{v}^\top [\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) \delta_y - \nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z}) \delta_y]}_{\text{Diff}_1} \\
&+ (1-\lambda) \underbrace{\mathbf{v}^\top [\nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z}) - \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})]}_{\text{Diff}_2}
\end{aligned}
\tag{5.23}
$$

Then by plugging the definition of $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})$ into the formula Diff_1 above and utilizing the Cauchy mean value theorem afterwards, the following formula could be derived:

$$
\begin{aligned}
&\text{Diff}_1 = \mathbf{v}^\top [-[\nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}^{(0)}, \mathbf{x}))] \\
&, -[\nabla_{\mathbf{w}} \log(p^{(2)}(\mathbf{w}^{(k)}, \mathbf{x})) - \nabla_{\mathbf{w}} \log(p^{(2)}(\mathbf{w}^{(0)}, \mathbf{x}))]] \delta_y \\
&= \mathbf{v}^\top [\int_0^1 -\nabla^2_{\mathbf{w}} \log(p^{(0)}(\mathbf{w}, \mathbf{x}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds (\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \\
&, \int_0^1 -\nabla^2_{\mathbf{w}} \log(p^{(1)}(\mathbf{w}, \mathbf{x}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds (\mathbf{w}^{(k)} - \mathbf{w}^{(0)})] \delta_y \\
&= \mathbf{v}^\top [\mathbf{H}^{(0)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \delta_{y,0} + \mathbf{H}^{(1)}(\mathbf{w}^{(k)}, \mathbf{z})(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \delta_{y,1}]
\end{aligned}
\tag{5.24}
$$

Note that for the binary classification problem, $\delta_y = \text{onehot}(c) - \tilde{y} = \text{onehot}(c) - [\tilde{y}_0, \tilde{y}_1], c = 0, 1$, which is equivalent to $[-\tilde{y}_0, 1-\tilde{y}_1]$ or $[1-\tilde{y}_0, -\tilde{y}_1]$ (depending on the value of c). Recall that $\tilde{y}_0$ and $\tilde{y}_1$ represent the probability that the sample $\mathbf{z}$ belongs to the class 0 and class 1 respectively, which satisfies $\tilde{y}_0 + \tilde{y}_1 = 1$. Then δ_y can be further transformed into the following formula:

$$
\delta_y = \text{onehot}(c) - \tilde{y} = \begin{cases} [-\tilde{y}_0, 1-\tilde{y}_1] = [-\tilde{y}_0, \tilde{y}_0], & \text{c=0} \\ [1-\tilde{y}_0, -\tilde{y}_1] = [\tilde{y}_1, -\tilde{y}_1], & \text{c=1} \end{cases}
$$

The above formula tells us that the two entries in the vector δ_y have the same magnitude

but have different symbols. Therefore, I can rewrite δ_y as $\delta_y = [-\delta, \delta], \delta = \tilde{y}_0$ or $-\tilde{y}_1$, which can be plugged into Equation (5.24), leading to the following results:

$$\text{Diff}_1 = \mathbf{v}^\top \delta[\mathbf{H}^{(1)}(\mathbf{w}^{(k)}, \mathbf{z}) - \mathbf{H}^{(0)}(\mathbf{w}^{(k)}, \mathbf{z})](\mathbf{w}^{(k)} - \mathbf{w}^{(0)}) \tag{5.25}$$

Recall that here I am focusing on the binary logistic regression model, meaning that I can explicitly derive the closed-form of $\mathbf{H}^{(0)}(\mathbf{w}^{(k)}, \mathbf{z})$ and $\mathbf{H}^{(1)}(\mathbf{w}^{(k)}, \mathbf{z})$. Specifically, for the binary logistic regression model, Equation (5.6), i.e. the loss on each sample, can be instantiated as:

$$\begin{aligned} F(\mathbf{w}, \tilde{\mathbf{z}}) &= -\sum\nolimits_{c=1}^{2} \tilde{y}^{(c-1)} \log(p^{(c-1)}(\mathbf{w}, \tilde{\mathbf{x}})) = -\sum\nolimits_{c=1}^{2} \tilde{y}^{(c-1)} \log(\text{softmax}^{(c-1)}(\mathbf{w}^\top \tilde{\mathbf{x}})) \\ &= -\tilde{y}^{(0)} \log(\frac{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}) - \tilde{y}^{(1)} \log(\frac{\exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}), \end{aligned}$$

in which $p^{(c-1)}(\mathbf{w}, \tilde{\mathbf{x}}) = \text{softmax}^{(c-1)}(\mathbf{w}^\top \tilde{\mathbf{x}})$ and I can split the model parameter $\mathbf{w}$ into two parts, i.e., $\mathbf{w}_0$ and $\mathbf{w}_1$, i.e. the model parameter corresponding to the class 0 and the class 1. Then by utilizing the definition of $\mathbf{H}^{(c)}(\mathbf{w}^{(k)}, \mathbf{z}), c = 0, 1$, it can be derived as below for the logistic regression model:

$$\begin{aligned} &\mathbf{H}^{(c)}(\mathbf{w}^{(k)}, \mathbf{z}) = -\int_{s=0}^{1} \nabla_{\mathbf{w}}^2 \log(p^{(c)}(\mathbf{w}, \mathbf{x}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds \\ &= -\int_{s=0}^{1} \nabla_{\mathbf{w}}^2 \log(\text{softmax}^{(c)}(\mathbf{w}))|_{\mathbf{w}=\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)})} ds \\ &= -\int_{s=0}^{1} \nabla_{\mathbf{w}}^2 \log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}})|_{\mathbf{w}=[\mathbf{w}_0, \mathbf{w}_1]=\text{vec}(\mathbf{w}^{(0)}+s(\mathbf{w}^{(k)}-\mathbf{w}^{(0)}))} ds \end{aligned}$$

In the above formula, first of all, I can explicitly derive the second derivative of $log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}})$ with respect to $\mathbf{w}$, which is calculated for $\mathbf{w}_0$ and $\mathbf{w}_1$ respectively

(due to $\mathbf{w} = [\mathbf{w}_0, \mathbf{w}_1]$), i.e.:

$$\nabla^2_{\mathbf{w}_0} \log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}) = \frac{-\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}} - \mathbf{w}_1^\top \tilde{\mathbf{x}}} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top}{(\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}})^2}$$

$$\nabla^2_{\mathbf{w}_1} \log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}) = \frac{-\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}} - \mathbf{w}_1^\top \tilde{\mathbf{x}}} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top}{(\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}})^2}$$

$$\nabla_{\mathbf{w}_0} \nabla_{\mathbf{w}_1} \log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}}) = \frac{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}} - \mathbf{w}_1^\top \tilde{\mathbf{x}}} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top}{(\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}})^2}$$

which is surprisingly free of c, meaning that $\nabla^2_{\mathbf{w}} \log(\frac{\exp^{-\mathbf{w}_c^\top \tilde{\mathbf{x}}}}{\exp^{-\mathbf{w}_0^\top \tilde{\mathbf{x}}} + \exp^{-\mathbf{w}_1^\top \tilde{\mathbf{x}}}})$ leads to the same results for both $c = 0$ and $c = 1$.

Therefore, I can conclude that $\mathbf{H}^{(0)}(\mathbf{w}^{(k)}, \mathbf{z}) = \mathbf{H}^{(1)}(\mathbf{w}^{(k)}, \mathbf{z})$, which can be further plugged into Equation (5.25), resulting in:

$$\text{Diff}_1 = 0$$

This thus indicates that:

$$(-\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)) = \text{Diff}_2.$$

Then by following similar analysis on the bound on Diff_2 in the proof for Theorem 13, I can derive the final result in Lemma 4.

□

To reduce the computational overhead, the integrated Hessian matrices, $\int_0^1 \mathbf{H}(\mathbf{w}^{(0)} + s(\mathbf{w}^{(k)} - \mathbf{w}^{(0)}), \tilde{\mathbf{z}})ds$ and $\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \tilde{\mathbf{z}})$, are approximated by their counterparts evaluated at $\mathbf{w}^{(0)}$, i.e., $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$ and $-\nabla^2_{\mathbf{w}} \log(p^{(j)}(\mathbf{w}^{(0)}, \tilde{\mathbf{x}}))$. As a consequence, the bounds can be calculated by applying several linear algebraic operations on $\mathbf{v}$, $\mathbf{w}^{(k)}$, $\mathbf{w}^{(0)}$ and some pre-computed formulas, i.e., the norm of the Hessian matrices, $\| - \nabla^2_{\mathbf{w}} \log(p^{(j)}(\mathbf{w}^{(0)}, \tilde{\mathbf{x}}))\|$ and $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$, and the gradients, $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$ and $\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$, which can be

computed as "provenance" information in the initialization step. Note that pre-computing $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$ and $\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$ is quite straightforward by leveraging Equation (5.7). Then the remaining question is how to compute $\| - \nabla^2_{\mathbf{w}} \log(p^{(j)}(\mathbf{w}^{(0)}, \tilde{\mathbf{x}}))\|$ and $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$ efficiently without explicitly evaluating the Hessian matrices. Since those two terms calculate the norm of one Hessian matrix, I therefore only take one of them as a running example to describe how to compute them in a feasible way, as shown below.

Pre-computing $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$ Since 1) a Hessian matrix is symmetric (due to its positive definiteness); and 2) the L2-norm of a symmetric matrix is equivalent to its eigenvalue with the largest magnitude [151], the L2 norm of one Hessian matrix is thus equivalent to its largest eigenvalue. To evaluate this eigenvalue, I use the Power method [152], which is shown in Algorithm 13.

Algorithm 13: Pre-compute $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$ in the initialization step

Input : A training sample $\mathbf{z} \in \mathcal{Z}$, the class j and the model parameter obtained in the initialization step: $\mathbf{w}^{(0)}$

Output: $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$

Initialize $\mathbf{g}$ as a random vector;

// Power method below

while $\boldsymbol{g}$ *is not converged* **do**

  Calculate $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}$ by using the auto-differentiation package

  Update $\mathbf{g}$: $\mathbf{g} = \frac{\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}}{\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}\|}$

end

Calculate the largest eigenvalue of $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$ in magnitude by using $\frac{\mathbf{g}^\top \mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}}{\|\mathbf{g}\|}$, which is equivalent to $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$. **return** $\|\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\|$.

Note that the algorithm above relies on the auto-differentiation package for calculating the Hessian-vector product effectively. Specifically, for a Hessian-vector product $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}$, it can be further rewritten as follows:

$$\begin{aligned} &\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g} = \nabla^2_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g} = \nabla_{\mathbf{w}}(\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}}))\mathbf{g} \\ &= \nabla_{\mathbf{w}}(\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}) \end{aligned} \tag{5.26}$$

in which the first equality utilizes the definition of the Hessian matrix while the last equality regards the vector $\mathbf{g}$ as a constant with respect to $\mathbf{w}$ and utilizes the chain rule in

reverse. Therefore, to obtain the result of $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}$, I can invoke the auto-differentiation package twice. The first one is on the loss $F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$, resulting in the first order derivative $\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})$, while the second one is on the product $\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}$, leading to the final result of $\mathbf{H}(\mathbf{w}^{(0)}, \tilde{\mathbf{z}})\mathbf{g}$.

Time complexity of Increm-INFL According to Theorem 13, evaluating the bound on $\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma)$ requires four major steps, including 1) computing the Hessian-vector product, $\mathbf{v}$, by employing the solution shown in Equation (5.26), which can be computed once for all training samples (suppose the time complexity of this step is $O(v)$; 2) computing $\mathbf{v}[\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})\delta_y + (1-\gamma)\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})]$ in $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ with two matrix-vector multiplications (recall that $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})$ and $\nabla_{\mathbf{w}} F(\mathbf{w}^{(0)}, \mathbf{z})$ are pre-computed), which requires $O(Cm)$ operations (m is used to denote the dimension of $\mathbf{w}$); 3) computing $\mathbf{v}(\mathbf{w}^{(k)} - \mathbf{w}^{(0)})$ and $\|\mathbf{v}\|\|\mathbf{w}^{(k)} - \mathbf{w}^{(0)}\|$, which requires $O(m)$ operations; 4) computing $\sum_{r=1}^{C} |\delta_{y,r}| \|\mathbf{H}^{(r)}(\mathbf{w}^{(k)}, \mathbf{z})\|$ and $\sum_{r=1}^{C} \delta_{y,r} \|\mathbf{H}^{(r)}(\mathbf{w}^{(k)}, \mathbf{z})\|$, which requires $O(C)$ operations (recall that $\|\mathbf{H}^{(r)}(\mathbf{w}^{(k)}, \mathbf{z})\|$ is also pre-computed). Hence, the overall overhead of evaluating the bound on $\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma)$ for all N training samples and all possible C classes is $O(v) + NC(O(Cm) + O(m) + O(C))$.

Suppose after Algorithm 14 is invoked, $n(\ll N)$ samples become the candidate influential training samples. Then the next step is to evaluate Equation (5.4) on each of those candidate samples for each possible deterministic class. Note that the main overhead of each invocation of Equation (5.4) comes from deriving the class-wise gradient $\nabla_y \nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})$ and the sample-wise gradient $\nabla_{\mathbf{w}} F(\mathbf{w}^{(k)}, \mathbf{z})$, which is supposed to have time complexity $O(\text{Grad})$. Therefore, the total time complexity of utilizing the Algorithm 14 first and evaluating Equation (5.4) on n candidate training samples afterwards is $O(v) + NC(O(Cm) + O(m) + O(C)) + ncO(\text{Grad})$. In contrast, without utilizing Algorithm 14, it is essential to evaluate Equation (5.4) on every training sample which thus requires $O(v) + NC \cdot O(\text{Grad})$ operations. Considering the fact that the time overhead of single gradient computation is much larger than $O(Cm)$, $O(m)$ or $O(C)$, then I can expect that with small n, Increm-INFL can lead to significant speed-ups.

Algorithm 14: Increm-INFL

Input : The number of samples to be cleaned at the k_{th} round: b

Output: A set of prioritized training samples for cleaning: $\mathcal{Z}_{inf}^{(k)}$

1 Initialize $\mathcal{Z}_{inf}^{(k)} = \{\}$

2 Calculate $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ and the perturbation bound on this term by using Theorem 13 for each uncleaned sample, $\tilde{\mathbf{z}} = (\tilde{\mathbf{x}}, \tilde{y})$, and each label perturbation, $\delta_y = \tilde{y} - onehot(c), (c = 0, 1, \ldots, C-1)$ Add the training samples producing Top-b smallest $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ to $\mathcal{Z}_{inf}^{(k)}$

3 obtain the largest perturbation upper bound, L, for all Top-b smallest $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$

4 For any remaining training sample, $\tilde{\mathbf{z}}$, if its lower perturbation bound of $\mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ is smaller than L with a certain δ_y, add it to $\mathcal{Z}_{inf}^{(k)}$ **return** $\mathcal{Z}_{inf}^{(k)}$

5.2.2 The model constructor phase (DeltaGrad-L)

At the k_{th} round of loop (2), after the human annotators clean the labels for a set of b influential training samples, $\mathcal{R}^{(k)}$, provided by the Sample selector, the next step is to update the model parameters in the Model constructor. To reduce the overhead of this step, I can regard the process of updating labels as two sub-processes, i.e. the deletions of the training samples, $\mathcal{R}^{(k)}$ (with the associated weight, γ), and the additions of those training samples with the cleaned labels (with the updated weight, 1), thus facilitating the use of DeltaGrad in this scenario. Specifically, the following modifications to Algorithm 10 are required: 1) the input deleted training samples should be $\mathcal{R}^{(k)}$; 2) the input cached model parameters and the corresponding gradients become the ones obtained at the $k-1_{st}$ round of the loop (2); 3) in line 3, instead of randomly sampling a mini-batch $\mathcal{A}_t$ from the added training samples $\mathcal{A}$, $\mathcal{A}_t$ should be replaced with the removed training samples from $\mathcal{B}_t$, i.e., $\mathcal{B}_t \bigcap \mathcal{R}^{(k)}$, but with updated labels; 4) the cleaned training samples and the uncleaned training samples so far are weighted by 1 and γ respectively (this only slightly modifies how the loss is calculated for each mini-batch).

5.2.3 The human annotation phase

As discussed in the beginning of this chapter, the Sample selector not only suggests which samples should be cleaned, but also suggests the candidate cleaned labels, which can be

regarded as one independent label annotator. When multiple annotators exist, I aggregate their labels by using majority vote to resolve possible label conflicts.

5.3 Experiments

I conducted extensive experiments in Python 3.6 and PyTorch 1.7.0 [102]. All experiments were conducted on a Linux server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 3 GeForce 2080 Titan RTX GPUs and 754GB of main memory. Code has been released in GitHub[2].

5.3.1 Experimental setup

In this section, I present necessary details on how to set up the experiments.

5.3.1.1 Datasets

Two types of datasets are used, one of which is annotated with ground-truth labels but no human generated labels, while the other is fully annotated with crowdsourced labels but only partially annotated by ground-truth labels. The former type (referred to as *Fully clean datasets*) is used to evaluate the quality of labels suggested by Infl by comparing them against the ground-truth labels. The latter type (referred to as *Crowdsourced datasets*) is used for evaluating the performance of our methods in more realistic settings.

Fully clean datasets: Three real medical image datasets are used: MIMIC-CXR-JPG (MIMIC for short) [153], Chexpert [41] and Diabetic Retinopathy Detection (Retina for short) [154]. The datasets are used to identify whether one or more diseases or findings exist for each image sample. In the experiments, I am interested in predicting the existence of findings called "Lung Opacity", "the referable Diabetic Retinopathy" and "Cardiomegaly" for MIMIC, Retina and Chexpert, respectively. The detailed descriptions of the three datasets are given below.

[2]see https://github.com/thuwuyinjun/Chef

MIMIC dataset is a large chest radiograph dataset containing 377,110 images, which have been partitioned into training set, validation set and test set. There are 13 binary labels for each image, corresponding to the existence of 13 different findings. Those labels are automatically extracted from the text [155], thus leading to possibly undetermined labels for some finds. In the experiments, I focused on predicting whether the finding "Lung Opacity" exists for each image and only retained those training samples with determined binary labels for this finding, eventually producing 85046 samples, 579 samples and 1628 samples in the training set, validation set and test set respectively.

Chexpert dataset is another large chest radiograph dataset consisting of 223,415 X-ray images as the training set and another 234 images as the validation set. Since the test set is not publicly available yet, I regard the original validation set as the test set and randomly selected 10% of the training samples as the validation set. This dataset is used to predict whether each of the 14 observations exists in each X-ray image. In the experiments, I focus on predicting the existence of the observation "Cardiomegaly" in each image. Similar to the pre-processing operations on MIMIC, I removed the training samples and the validation samples with undetermined labels (labeled as -1) for this observation, leading to 38629 samples and 4251 samples in the training set and validation set respectively. All the test samples, i.e. the original validation samples, are fully labeled, which are all retained in the experiments.

Retina dataset is an image dataset consisting of fully labeled retinal fundus photographs [154]. The target use of this dataset is to diagnose one eye disease called Diabetic Retinopathy (DR) for each image, which is classified into 5 categories based on severity. I followed [156] to predict whether an image belongs to a referable DR, which regard the label 1 and 2 as the referable one and the label 3-5 as the non-referable one. As a consequence, the original five-class classification problem is transformed into a binary classification problem. In the original version of Retina dataset, there are 35127 samples and 53576 samples in the training set and test. I randomly select 10% of the training samples as the validation samples and use the rest of them as the training set in the experiments.

Crowdsourced datasets: Three realistic crowdsourced datasets are used: Fashion 10000 (Fashion for short)[3] [157], Fact Evaluation Judgement (Fact for short)[4], and Twitter sentiment analysis (Twitter for short)[5]. Only a small portion of samples in the datasets have ground-truth labels while the rest are labeled by crowdsourcing workers (e.g., the labels of the Fashion dataset are collected through the Amazon Mechanical Turk (AMT) crowdsourcing platform). The Fashion dataset is an image dataset for distinguishing fashionable images from unfashionable ones; the Fact dataset uses RDF triples to represent facts about public figures and the classification task is to judge whether or not each fact is true; and the Twitter dataset consists of plain-text tweets on major US airlines for sentiment analysis, i.e., identifying positive or negative tweets. For the Fashion and Fact datasets, extra text is also associated with each sample, e.g. user comments on each image in Fashion and the evidence for each fact in Fact, which is critical for producing probabilistic labels (see Section 5.3.1.3). I further describe the three datasets in *Crowdsourced datasets* in details below.

Fashion dataset includes 30525 images and the label of each image represents whether it is fashionable or not, annotated by three different human annotators. In addition to those labels, some text information such as the users' comments is also associated with each image. However, ground-truth labels are not available in this dataset and simulated with the labels by aggregating the human annotated labels through majority vote. For the experiments in Section 5.3, similar to *Fully clean datasets*, I apply ResNet50 for feature transformation and run logistic regression model afterwards.

Fact dataset Each sample in Fact dataset is an RDF triple for representing one fact and there are over 40000 of such facts. Each such fact is labeled as true, false or ambiguous by five different human annotators. But the total number of human annotators is 57. Among all the samples, only 577 samples have ground-truth labels. In the experiments, I removed the samples with the ground-truth label "ambiguous" and randomly partition the remaining samples with ground-truth labels into two parts, Although there are three different labels,

[3]available at `http://skuld.cs.umass.edu/traces/mmsys/2014/user05.tar`

[4]available at `https://sites.google.com/site/crowdscale2013/shared-task/task-fact-eval`

[5]available at `https://github.com/naimulhuq/Capstone/blob/master/Data/Airline-Full-Non-Ag-DFE-Sentiment%20(raw%20data).csv`

I ignore the label 'ambiguous', meaning that I only conduct a binary classification task on this dataset. However, it is likely that the aggregated label for some uncleaned training sample becomes 'ambiguous' even after I resolve the labeling conflicts between different human annotators. To deal with this, the probabilistic labels of this sample is not updated for representing the labeling uncertainties from the human annotators.

To facilitate the feature transformation as mentioned in Section 5.3, I concatenate each RDF triple as one sentence and then employ the pre-trained bert-based transformer [158] for transforming each raw text sample into a sequence of embedding vectors. To guarantee batch training on this dataset, only the last 20 embedding vectors are used. If the total number of embedding sequence for a sample is smaller than 20, I pad this sequence with zero vectors. As introduced in Section 5.3, to identify whether a fact is true or not, it is essential to compare this RDF triple against the associated evidence (represented by a sentence). Therefore, by following the above principle, I transform each piece of evidence into a embedding sequence and trim the length of this sequence to 20 for accelerating the training process.

Twitter dataset is comprised of ∼12k tweets for sentimental analysis. In other words, the classification problem on this dataset is to judge whether the expression in each tweet is positive, negative or neutral. The labels of those samples are provided by a group of 507 human annotators and each individual tweet is labeled by three different human annotators. Among all the samples, 577 of them have ground-truth labels. Similar to Fact dataset, only the positive label and the negative label are employed in the experiments. Therefore, the samples taking the neutral label as the ground truth are removed. Also, if the aggregated human annotated labels on one uncleaned sample is neural, then the probabilistic label on this sample is not updated Plus, I generate a 768-D embedding sequence by running the pre-trained bert-based transformer on each tweet and trim the length of the resulting embedding sequence to 20. When logistic regression model is used,

5.3.1.2 Partition training-validation-test sets

Prior to initiating the model training process, one necessary setup is to produce appropriate training, validation and test sets for the above six datasets, which are described below.

Since some samples in the datasets have missing or unknown ground-truth labels, I remove them in the experiments. Also, except for MIMIC which has 579 validation samples and 1628 test samples, other datasets do not have well-defined validation and test set. For example, as of the time the experiments were performed, the test samples of Chexpert had not been released. To remedy this, I partition the Chexpert validation set into two parts to create validation and test sets, each of which have 234 samples. Since there was no validation set for Retina, I randomly select roughly 10% training samples, i.e., 3512 samples, as the validation set. Similarly, for the Twitter and Fact datasets, I randomly partition the set of samples with ground-truth labels as the validation set and test set, and regard all the other samples as the training set. Since ground-truth labels are not available in the Fashion dataset, I randomly select roughly 0.5%[6] of the sample samples as the validation set and test set, each containing 146 samples. The "ground-truth" labels for those samples are determined by aggregating human annotated labels using majority vote. The remaining samples in this dataset are then regarded as training samples. In the end, the six datasets, i.e., MIMIC, Retina, Chexpert, Fashion, Fact and Twitter include ∼78k,∼31k,∼38k,∼29k,∼38k and ∼12k training samples. More detailed statistics of the six datasets are given in Table 5.1.

Table 5.1 – Sizes of *Fully clean datasets* and *Crowdsourced datasets*

Dataset	MIMIC	Retina	Chexpert	Fashion	Fact	Twitter
Training set	78487	31615	37882	29031	38176	11606
Validation set	579	3512	234	146	255	37
Test set	1628	53576	234	146	259	37
# of samples with ground truth	80649	88703	43114	29323	514	74

[6]This ratio is determined based on the observation that in the Twitter and Fact datasets, the percentage of samples with ground-truth labels is less than 1% of the size of the entire dataset.

5.3.1.3 Producing probabilistic labels

Due to the lack of probabilistic labels or labeling functions [35] for the datasets, I leverage [48],[159] or [160] to *automatically* derive suitable labeling functions and thus probabilistic labels in the experiments. Note that [48] and [159] deal with text data (including the text associated with image data) while [160] targets pure image data. However, the time and space complexity of [160] is quadratic in the dataset size, and does not scale to large image datasets such as our *Fully clean datasets*. Furthermore, no text information is available for images in *Fully clean datasets*, so it is not feasible to use [48] or [159]. As a result, random probabilistic labels are produced for all training samples. For *Crowdsourced datasets*, I apply [48] on the extra text information in Fashion (e.g. user comments for each image) and the plain-text tweets in the Twitter dataset to produce probabilistic labels. For the Fact dataset, the two texts for each sample (i.e. the RDF triples and the associated evidence) are compared using [161] to generate labeling functions.

5.3.1.4 Human annotator setup

For *Crowdsourced datasets*, I can use the crowdsourced labels as the cleaned labels for the uncleaned training samples. However, no such labels are available in *Fully clean datasets*. To remedy this, I note that the error rate of manually labeling medical images is typically between 3% and 5%, but sometimes can be up to 30% [162]. I therefore produce synthetic human annotated labels by flipping the ground truth labels of a randomly selected 5% of the samples [7]. I assume three independent annotators, and aggregate their labels as the cleaned labels using majority vote (denoted INFL (one)). Since Infl and DUTI [115] can suggest cleaned labels, those labels can be used as cleaned labels by themselves for the uncleaned samples (denoted INFL (two)) or be combined with two other simulated human annotators for label cleaning (denoted INFL (three)).

[7] Recall that although the samples have probabilistic labels, their true labels are known by construction.

5.3.1.5 Model constructor setup

Throughout the paper I assume that strong convexity holds on the ML models. Therefore, in this section, to justify the performance advantage of our design as a whole (including Increm-INFL, DeltaGrad-L and Infl), I focus on a scenario where pre-trained models are leveraged for feature transformation and then a logistic regression model is used for classification, which has emerged as a convention for medical image classification tasks [156]. Specifically, in the experiments, I use a pre-trained ResNet50 [163] for the image datasets (*Fully clean datasets* and Fashion), and use a pre-trained BERT-based transformer [158] for the text datasets (Fact and Twitter). Unless explicitly stated, stochastic gradient descent (SGD) is used in the subsequent training process with the default mini-batch size of 2000, and the default weight $\gamma = 0.8$ on the uncleaned samples. Early stopping is also applied to avoid overfitting. Other hyper-parameters for model training are included in Table 5.2.

Table 5.2 – The hyper-parameters for each dataset

Dataset	MIMIC	Retina	Chexpert	Fashion	Fact	Twitter
Learning rate	0.0005	0.001	0.02	0.05	0.005	0.01
L2 regularization	0.05	0.05	0.05	0.001	0.01	0.01
# of epochs	150	200	200	200	150	400

As discussed in the beginning of this chapter, other than the initialization step, I can construct the models by either retraining from scratch (denoted Retrain) or leveraging DeltaGrad for incremental updates. As indicated in Algorithm 10, it is essential to specify three hyper-parameters for DeltaGrad, i.e., the period T_0, the history size m_0 and the "burn-in" iteration number j_0, which are configured as $m_0 = 2$, $j_0 = 10$ and $T_0 = 10$ in the experiments.

However, the strong convexity assumption on model type is only required for Increm-INFL and DeltaGrad-L, but not for Infl. Hence, extra experiments are conducted using convolutional neural networks (CNNs), which are presented in the Section 5.3.3.1. The results also demonstrate the performance advantage of Infl in more general settings.

5.3.1.6 Sample selector setup

I assume that the clean budge $B = 100$, meaning that 100 training samples are cleaned in total. I further vary the number of samples to be cleaned at each round, i.e. the value of b.

5.3.1.7 Baseline methods

Baseline against Infl I compare Infl against several baseline methods, including other versions of the influence function, i.e. Equation (5.2) [44] (denoted by Infl-D, which is also INFL in Chapter 4) and Equation (5.5) [115] (denoted by Infl-Y) and DUTI. Since solving the bi-level optimization problem in DUTI is extremely expensive, I only run DUTI once to identify the Top-100 influential training samples.

Since active learning and noisy sample detection algorithms can prioritize the most influential samples for label cleaning, they are also compared against Infl. Specifically, I consider two active learning methods, i.e., least confidence based sampling method (denoted by Active (one)) and entropy based sampling method (denoted by Active (two)) [45], and two noisy sample detection algorithms, i.e., O2U [46] and TARS [111].

Note that many of these baseline methods are not applicable in the presence of probabilistic labels and regularization on uncleaned training samples. Hence, I modify the methods to handle these scenarios or adjust the experimental set-up to create a fair comparison. For example, at the end of this section, I present necessary modifications to DUTI so that it can handle probabilistic labels. However, it is not straightforward to modify DUTI for quantifying the effect of up-weighting the training samples after they are cleaned. I therefore only compare DUTI against Infl when all the training samples are equally weighted (i.e. $\gamma = 1$ in Equation (5.1)) Similarly, TARS is only applicable when the noisy labels are either 0 or 1 rather than probabilistic ones. Therefore, to compare Infl and TARS, I round the probabilistic labels to their nearest deterministic labels for a fair comparison. For other baseline methods such as Active (one), Active (two), O2U and Infl-D, no modifications are made other than using Equation (5.1) for model training.

Baseline against DeltaGrad-L and Increm-INFL Recall that DeltaGrad-L incre-

mentally updates the model after some training samples are cleaned. I compare this with retraining the model from scratch (denoted as Retrain). I also compare the running time for selecting the influential training samples with and without Increm-INFL. When Increm-INFL is not used, it is denoted as Full.

Adapting DUTI to handle probabilistic labels According to [115], the original version of DUTI is as follows:

$$\begin{aligned}&\min_{\mathbf{Y}'=[y'_1,y'_2,\ldots,y'_n],\hat{\mathbf{w}}} [\frac{1}{|\mathcal{Z}_{\text{val}}|}\sum_{\mathbf{z}\in\mathcal{Z}_{\text{val}}} F(\hat{\mathbf{w}},\mathbf{z}) + \frac{1}{n}\sum_{i=1}^{n} F(\hat{\mathbf{w}},(\mathbf{x},y'_i)) + \frac{\gamma}{n}\sum_{i=1}^{n}(1-y'_{i,y_i})],\\ &\text{s.t. } \hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}}\frac{1}{n}\sum_{i=1}^{n} F(\mathbf{w},(\mathbf{x}_i,y'_i))\end{aligned} \tag{5.27}$$

which is defined on the training dataset $\mathcal{Z} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ and the validation dataset $\mathcal{Z}_{\text{val}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{Z}_{\text{val}}|}$. In the formula above, each y'_i is a vector of length C (recall that C represents the number of classes) and the term y'_{i,y_i} indicates the $(y_i)_{th}$ entry in the vector y'_i, which implicitly suggests that each y_i should be a deterministic label.

Note that if y_i is a probabilistic label (represented by a probabilistic vector of length C), I cannot calculate the term y'_{i,y_i}. Therefore, I replace y_i in y'_{i,y_i} by using the index with the largest entry in y_i.

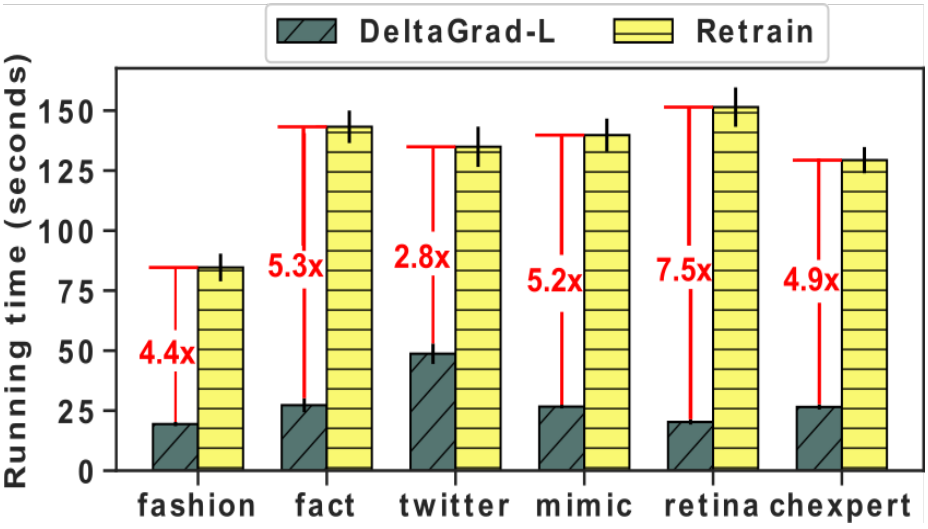


Figure 5.2 – Comparison of accumulated running time between DeltaGrad-L and Retrain

Table 5.3 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 100, \gamma = 0.8$)

	MIMIC	Retina	Chexpert
uncleaned	0.6284±0.0012	0.5565±0.0019	0.5244±0.0016
Infl-D	0.6283±0.0011	0.5556±0.0012	0.5244±0.0033
Active (one)	0.6286±0.0008	0.5566±0.0029	0.5248±0.0024
Active (two)	0.6286±0.0008	0.5566±0.0029	0.5248±0.0024
O2U	0.1850±0.0006	0.1331±0.0012	0.5276±0.0012
INFL (one)	0.6292±0.0005	0.5580±0.0013	0.5286±0.0023
INFL (two)	**0.6293±0.0012**	**0.5582±0.0011**	**0.5297±0.0022**
INFL (three)	0.6293±0.0008	0.5581±0.0009	0.5289±0.0022

Table 5.4 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 100, \gamma = 0.8$)

	Fashion	Fact	Twitter
uncleaned	0.5140±0.0142	0.6595±0.0017	0.6485±0.0050
Infl-D	0.5143±0.0146	0.6596±0.0018	0.6530±0.0088
Active (one)	0.5145±0.0144	0.6598±0.0017	0.6540±0.0045
Active (two)	0.5145±0.0144	0.6598±0.0017	0.6540±0.0045
O2U	0.5148±0.0142	0.6599±0.0014	0.6481±0.0023
INFL (one)	**0.5178±0.0132**	0.6601±0.0021	0.6594±0.0034
INFL (two)	0.5177±0.0132	**0.6609±0.0021**	**0.6680±0.0044**
INFL (three)	0.5177±0.0126	0.6603±0.0021	0.6594±0.0032

5.3.2 Experimental design

In this section, I design the following three experiments:

Table 5.5 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 10, \gamma = 0.8$)

	MIMIC	Retina	Chexpert
uncleaned	0.6284±0.0012	0.5565±0.0019	0.5244±0.0016
Infl-D	0.6283±0.0011	0.5556±0.0016	0.5246±0.0036
Active (one)	0.6287±0.0005	0.5568±0.0001	0.5246±0.0020
Active (two)	0.6287±0.0005	0.5568±0.0016	0.5246±0.0020
O2U	0.1850±0.0008	0.1314±0.0006	0.5281±0.0016
INFL (one)	0.6292±0.0007	0.5579±0.0013	0.5287±0.0024
INFL (two)	**0.6293±0.0011**	**0.5582±0.0003**	**0.5300±0.0024**
INFL (two)+ DeltaGrad	**0.6292±0.0005**	**0.5610±0.0010**	**0.5295±0.0030**
INFL (three)	0.6292±0.0008	0.5581±0.0018	0.5291±0.0023

Table 5.6 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 10, \gamma = 0.8$)

	Fashion	Fact	Twitter
uncleaned	0.5140±0.0142	0.6595±0.0017	0.6485±0.0050
Infl-D	0.5143±0.0144	0.6596±0.0018	0.6518±0.0081
Active (one)	0.5145±0.0135	0.6600±0.0017	0.6515±0.0082
Active (two)	0.5145±0.0135	0.6600±0.0017	0.6515±0.0082
O2U	0.5152±0.0143	0.6598±0.0010	0.6490±0.0067
INFL (one)	0.5178±0.0125	0.6601±0.0019	0.6578±0.0039
INFL (two)	**0.5181±0.0131**	**0.6609±0.0020**	**0.6697±0.0058**
INFL (two)+ DeltaGrad	**0.5195±0.0144**	**0.6609±0.0065**	**0.6597±0.0027**
INFL (three)	0.5180±0.0128	0.6602±0.0022	0.6586±0.0032

Table 5.7 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 100, \gamma = 1$)

	MIMIC	Retina	Chexpert
uncleaned	0.6310±0.0010	0.5547±0.0020	0.5360±0.0123
Infl-D	0.6310±0.0010	0.5543±0.0015	0.5354±0.0108
Infl-Y	0.6310±0.0011	0.5546±0.0019	0.5360±0.0118
DUTI	0.6310±0.0011	0.5558±0.0019	0.5361±0.0127
Active (one)	0.6314±0.0008	0.5554±0.0018	0.5366±0.0127
Active (two)	0.6314±0.0008	0.5554±0.0018	0.5366±0.0127
O2U	0.1278±0.0080	0.0940±0.0023	0.5282±0.0043
INFL (one)	0.6320±0.0011	0.5564±0.0020	0.5403±0.0122
INFL (two)	**0.6321±0.0011**	**0.5567±0.0021**	**0.5444±0.0080**
INFL (three)	**0.6321±0.0010**	0.5565±0.0021	0.5403±0.0131

Table 5.8 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 100, \gamma = 1$)

	Fashion	Fact	Twitter
uncleaned	0.5264±0.0078	0.6584±0.0015	0.7034±0.0062
Infl-D	0.5262±0.0076	0.6587±0.0016	0.6853±0.0140
Infl-Y	0.5265±0.0081	0.6584±0.0015	0.7102±0.0125
DUTI	0.5238±0.0070	0.6582±0.0056	0.6230±0.0149
Active (one)	0.5267±0.0041	0.6580±0.0020	0.7051±0.0089
Active (two)	0.5267±0.0041	0.6580±0.0020	0.7051±0.0089
O2U	0.5277±0.0065	0.6587±0.0006	0.6401±0.0175
INFL (one)	0.5297±0.0068	0.6586±0.0016	0.7164±0.0017
INFL (two)	**0.5301±0.0053**	**0.6588±0.0016**	**0.7349±0.0290**
INFL (three)	0.5299±0.0059	0.6585±0.0017	0.7200±0.0109

Table 5.9 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 10, \gamma = 1$)

	MIMIC	Retina	Chexpert
uncleaned	0.6310±0.0010	0.5547±0.0020	0.5360±0.0123
Infl-D	0.6310±0.0010	0.5543±0.0015	0.5355±0.0112
Infl-Y	0.6310±0.0011	0.5547±0.0018	0.5359±0.0121
Active (one)	0.6316±0.0007	0.5552±0.0018	0.5368±0.0127
Active (two)	0.6316±0.0007	0.5552±0.0018	0.5368±0.0127
O2U	0.1284±0.0077	0.0934±0.0021	0.5281±0.0036
INFL (one)	**0.6321±0.0010**	0.5564±0.0018	0.5402±0.0117
INFL (two)	**0.6321±0.0010**	**0.5567±0.0019**	**0.5412±0.0120**
INFL (three)	**0.6321±0.0010**	0.5564±0.0019	0.5406±0.0124

Table 5.10 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 10, \gamma = 1$)

	Fashion	Fact	Twitter
uncleaned	0.5264±0.0078	0.6584±0.0015	0.7034±0.0062
Infl-D	0.5263±0.0076	0.6585±0.0015	0.5854±0.0099
Infl-Y	0.5265±0.0081	0.6585±0.0017	0.6999±0.0020
Active (one)	0.5268±0.0041	0.6584±0.0021	0.7030±0.0109
Active (two)	0.5268±0.0041	0.6584±0.0021	0.7030±0.0109
O2U	0.5275±0.0065	0.6586±0.0020	0.6268±0.0069
INFL (one)	0.5298±0.0068	0.6585±0.0016	0.7153±0.0019
INFL (two)	**0.5303±0.0053**	**0.6588±0.0016**	**0.7420±0.0019**
INFL (three)	0.5300±0.0059	0.6585±0.0017	0.7171±0.0029

Table 5.11 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 100, \gamma = 0$)

	MIMIC	Retina	Chexpert
uncleaned	0.6323±0.0041	0.5614±0.0018	0.5231±0.0009
Infl-D	0.8880±0.0174	0.6728±0.0138	0.5709±0.0059
Active (one)	0.8641±0.0348	**0.6898±0.0038**	0.5752±0.0512
Active (two)	0.8641±0.0348	**0.6898±0.0038**	0.5752±0.0512
O2U	0.8933±0.0082	0.6707±0.0340	0.5825±0.0195
INFL (one)	0.8740±0.0306	0.6756±0.0133	0.5853±0.0238
INFL (two)	**0.8989±0.0008**	0.6894±0.0048	**0.5924±0.0171**
INFL (three)	0.8853±0.0196	0.6884±0.0077	0.5878±0.0188

Table 5.12 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 100, \gamma = 0$)

	Fashion	Fact	Twitter
uncleaned	0.5057±0.0163	0.6604±0.0020	0.6335±0.0301
Infl-D	0.5789±0.0437	0.6584±0.0221	0.6534±0.0248
Active (one)	0.5593±0.0089	0.6723±0.0185	0.6324±0.0024
Active (two)	0.5593±0.0089	0.6723±0.0185	0.6324±0.0112
O2U	0.5830±0.0021	0.6553±0.0232	**0.7703±0.0210**
INFL (one)	0.5905±0.0212	0.6618±0.0192	0.6029±0.0689
INFL (two)	**0.6020±0.0431**	**0.6691±0.0159**	0.6739±0.0403
INFL (three)	0.5975±0.0139	0.6616±0.0221	0.6232±0.0560

Table 5.13 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Fully clean datasets* ($b = 10, \gamma = 0$)

	MIMIC	Retina	Chexpert
uncleaned	0.6323±0.0041	0.5614±0.0018	0.5231±0.0009
Infl-D	**0.9013±0.0006**	0.6550±0.0613	0.5736±0.0100
Active (one)	0.9008±0.0007	0.6916±0.0038	0.5890±0.0190
Active (two)	0.9007±0.0007	0.6916±0.0038	0.5890±0.0190
O2U	0.9010±0.0008	**0.6962±0.0193**	0.5775±0.0271
INFL (one)	0.9007±0.0004	0.6902±0.0007	0.6275±0.0329
INFL (two)	0.8991±0.0012	0.6904±0.0254	**0.6382±0.0225**
INFL (three)	0.9008±0.0010	0.6802±0.0108	0.6254±0.0206

Table 5.14 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned on *Crowdsourced datasets* ($b = 10, \gamma = 0$)

	Fashion	Fact	Twitter
uncleaned	0.5057±0.0163	0.6604±0.0020	0.6335±0.0301
Infl-D	0.5956±0.0296	0.6511±0.0306	0.6140±0.0226
Active (one)	0.5651±0.0407	0.6532±0.0305	0.6891±0.0024
Active (two)	0.5651±0.0407	0.6532±0.0305	0.6891±0.0086
O2U	0.5498±0.0486	0.6829±0.0157	0.7547±0.0087
INFL (one)	0.6449±0.0126	0.6576±0.0283	0.7406±0.1798
INFL (two)	**0.6926±0.0148**	**0.6871±0.0801**	**0.8184±0.0446**
INFL (three)	0.6706±0.0174	0.6582±0.0229	0.7711±0.0258

Table 5.15 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned (against TARS, b=100)

	MIMIC	Chexpert	Retina	Fashion
uncleaned	0.6413±0.0008	0.5359±0.0040	0.5702±0.0015	0.6280±0.0035
Infl-D	0.6647±0.0129	0.5506±0.0015	0.6077±0.0090	0.6318±0.0047
Active (one)	0.6569±0.0240	0.5404±0.0025	0.5910±0.0028	0.6329±0.0070
Active (two)	0.6569±0.0240	0.5404±0.0025	0.5910±0.0028	0.6329±0.0070
O2U	**0.6686±0.0016**	0.5419±0.0008	0.6021±0.0110	0.6380±0.0027
TARS	0.6022±0.0016	0.5257±0.0031	0.5573±0.0295	0.4964±0.0022
INFL (one)	0.6606±0.0117	0.5505±0.0022	0.6015±0.0082	0.6365±0.0022
INFL (two)	0.6375±0.0445	**0.5671±0.0065**	**0.6168±0.0063**	**0.6460±0.0060**
INFL (three)	0.6600±0.0079	0.5537±0.0037	0.6057±0.0076	0.6372±0.0026

Table 5.16 – Comparison of the model prediction performance (F1 score) after 100 training samples are cleaned (against TARS, b=10)

	MIMIC	Chexpert	Retina	Fashion
uncleaned	0.6413±0.0008	0.5359±0.0040	0.5702±0.0015	0.6280±0.0035
Infl-D	0.6874±0.0110	0.5581±0.0119	0.6098±0.0111	0.6318±0.0047
Active (one)	0.6815±0.0063	0.5404±0.0136	0.5989±0.0062	0.6329±0.0070
Active (two)	0.6815±0.0063	0.5404±0.0136	0.5989±0.0062	0.6329±0.0070
O2U	0.6698±0.0019	0.5396±0.0045	0.6026±0.0012	0.6380±0.0027
TARS	0.6025±0.0008	0.5333±0.0029	0.5572±0.0296	0.4964±0.0022
INFL (one)	0.6867±0.0005	0.5626±0.0061	0.6136±0.0097	0.6365±0.0022
INFL (two)	**0.6968±0.0220**	**0.5863±0.0046**	**0.6347±0.0034**	**0.6460±0.0060**
INFL (three)	0.6925±0.0149	0.5579±0.0128	0.6173±0.0070	0.6372±0.0026

Table 5.17 – Comparison of the model prediction performance (average F1 score) after 100 training samples are cleaned (CNN, b = 100)

	uncleaned	Infl-D	Active (one)	Active (two)	O2U	INFL (one)	INFL (two)	INFL (three)
MIMIC	0.6301	0.6563	0.6569	0.6569	0.6616	0.6598	**0.6642**	0.6614
Retina	0.5539	0.5697	0.5698	0.5698	0.5707	0.5700	**0.5732**	0.5701
Fact	0.6883	0.7560	0.7566	0.7516	0.7739	0.7553	**0.8063**	0.7654
Twitter	0.6697	0.6707	0.6670	0.6670	0.6763	0.6903	**0.6980**	0.6897

Table 5.18 – Comparison of the model prediction performance (average F1 score) after 100 training samples are cleaned (CNN, b = 10)

	uncleaned	Infl-D	Active (one)	Active (two)	O2U	INFL (one)	INFL (two)	INFL (three)
MIMIC	0.6301	0.6591	0.6601	0.6601	0.6593	0.6566	**0.6646**	0.6593
Retina	0.5539	0.5699	0.5699	0.5699	0.5695	0.5699	**0.5731**	0.5702
Fact	0.6883	0.7560	0.7566	0.7516	0.7739	0.7553	**0.8063**	0.7654
Twitter	0.6697	0.6712	0.6668	0.6668	0.6772	0.6901	0.6887	**0.6929**

5.3.2.1 Experiments for evaluating Infl

Exp1 In this experiment, I compared the model prediction performance after Infl and other baseline methods (including Infl-D, Active (one), Active (two), O2U) are applied to select 100 training samples for cleaning. Recall that there are three different strategies that Infl can use to provide cleaned labels and their performance is compared. To show the benefit of using a smaller batch size b, I choose two different values for b, i.e. 100 and 10. Since the ground-truth labels are available for all samples in *Fully clean datasets*, I count how many of them match the labels suggested by Infl. Also other than the default value of γ as 0.8, I choose two other different values, 0 and 1 for γ for more extensive comparisons between Infl and other baseline methods. As mentioned in Section 5.3.1.7, when $\gamma = 1$, meaning that all the training samples are equally weighted, I can also conduct fair comparison between Infl and DUTI.

Exp2 In this experiment, I aim at comparing Infl against TARS. As mentioned in Section 5.3.1.7, TARS is only applicable when the noisy labels are either 0 or 1 rather than probabilistic ones. Therefore, for a fair comparison between Infl and TARS, I round the probabilistic labels on the uncleaned training samples to their nearest deterministic labels. I notice that TARS may not be suitable for all the datasets. This is because to determine the

Table 5.19 – Running time of Increm-INFL and Full

	Time_{inf} (s)		Time_{grad} (s)	
	Full	Increm-INFL	Full	Increm-INFL
MIMIC	151.4±0.5	**2.77±0.03 (54.7x)**	145.4±0.7	**0.17±0.03 (855x)**
Retina	74.0±0.6	**1.36±0.04 (54.4x)**	70.8±0.6	**0.21±0.03 (337x)**
Chexpert	72.5±0.2	**17.9±1.9 (4.1x)**	69.3±0.2	**14.7±1.5 (4.7x)**
Fashion	66.4±3.6	**8.7±0.6 (7.6x)**	57.1±3.3	**0.81±0.07 (70.5x)**
Fact	73.8±4.0	**6.1±0.8 (12.1x)**	72.5±6.0	**4.7±0.1 (15.4x)**
Twitter	33.1±2.3	**14.1±0.4 (2.3x)**	30.2±1.1	**12.7±0.1 (2.4x)**

influence of each uncleaned training sample, TARS needs to estimate how each uncleaned label will be changed if it is to be cleaned. This depends on "all" the possible combinations of labels provided by "all" human annotators, which are thus exponential in the number of human annotators. Therefore, since the number of human annotators for Fact and Twitter dataset is not small (over 50), I only compare Infl against TARS on *Fully clean datasets* and Fashion dataset. In this experiment, I still train logistic regression models on the features transformed by using the pre-trained models.

Exp3 I also conduct some initial experiments when neural network models are used in the *Model constructor*. To goal is to compare Infl (with different strategies to clean labels) against all the baseline methods mentioned in Section 5.3 (including Infl-D, Active (one), Active (two), and O2U) in this more general setting. Specifically, for the image dataset, I applied the LeNet [164] (a classical convolutional neural network structure) on the original image features (instead of features transformed by using transfer learning). For the text dataset, such as Fact and Twitter dataset, similar to Section 5.3, I still transform each plain-text sample into the corresponding embedding representations by using the pre-trained bert-based transformer and then applied one 1D convolutional neural network on the transformed embedding representations. I found that the performance of applying LeNet model on Fashion and Chexpert dataset is significantly worse than that when the pre-trained models are used, even when all the ground-truth labels or the aggregated human annotated labels are used. Therefore, I only present the experimental results on MIMIC, Retina, Fact and Twitter dataset.

Exp4 Finally, I conduct an experiment to explore an appropriate b value which can bal-

ance the model performance and the running time given a fixed cleaning budget. Specifically, I set up the clean budget as 1000 and vary b from 10 to 1000. All the other hyper-parameters are the same as the above experimental design.

5.3.2.2 Experiments for evaluating Increm-INFL

Exp5 This experiment compares the running time of selecting the Top-b (with $b = 10$) influential training samples (denoted Time_{inf}) with and without using Increm-INFL at each round in the *Sample selector* phase. Recall that the most time-consuming step to evaluate Equation (5.4) is to compute the class-wise gradients for each sample and the sample-wise gradients. Therefore, its running time (denoted as Time_{grad}) is also recorded. For Increm-INFL, the time to compute the bounds in Theorem 13 is also included in Time_{inf}.

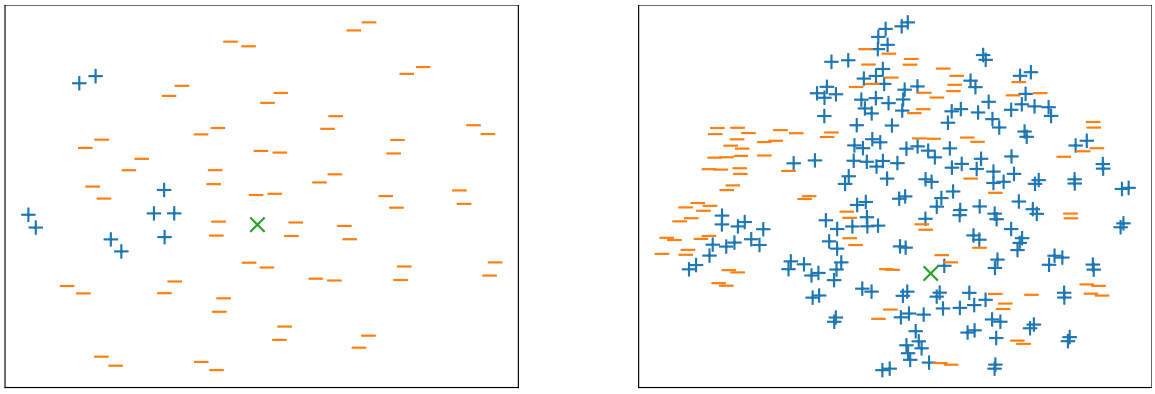
5.3.2.3 Experiments for evaluating DeltaGrad-L

Exp6 The main goal of this experiment is to explore the difference in running time between Retrain and DeltaGrad-L for updating the model parameters in the *Model constructor* phase. In addition, the model parameters produced by DeltaGrad-L and Retrain are not exactly the same [95], which could lead to different influence values for each training sample and thus produce different models in subsequent cleaning rounds. Therefore, I also explore whether such differences produce divergent prediction performance for DeltaGrad-L and Retrain.

5.3.3 Experimental results

5.3.3.1 Experiments for evaluating Infl

Exp1 Experimental results are given in Table 5.3-5.14. When $\gamma = 0.8$ or $\gamma = 1$, I observe that with fixed b, e.g., 10, INFL (two) performs best across almost all datasets. Recall that INFL (two) uses the derived labels produced by Infl as the cleaned labels without additional human annotated labels. Due to its superior performance, especially on *Crowdsourced datasets*, this implies that the quality of the labels provided by Infl could actually be better than that of the human annotated labels.



(a) Twitter (b) Fashion

Figure 5.3 – Visualization of the validation samples, test samples and the most influential training sample S ('+', '-' and 'X' denote the positive ground-truth samples, negative ground-truth samples and the sample S respectively)

To further understand the reason behind this, I compared the labels suggested by Infl against their ground-truth labels for *Fully clean datasets*. It turns out that over 70% are equivalent (89 for Retina, 79 for Chexpert and 95 for MIMIC). Note that even the ground-truth labels of these three datasets are not 100% accurate. In the Chexpert dataset, for example, the ground-truth labels are generated through an automate labeling tool rather than being labeled by human annotators, thereby leading to possible labeling errors. Those erroneous labels may not match the labels provided by Infl, thus leading to worse model performance (see the performance difference between INFL (one) and INFL (two) for Chexpert dataset).

However, the above comparison could not be done for *Crowdsourced datasets* due to the lack of ground-truth labels. I therefore investigate the relationship between the samples with ground-truth labels and the influential samples identified by Infl. Specifically, I use t-SNE [165] to visualize the samples with ground-truth labels for the Twitter and Fashion datasets after feature transformation using the pre-trained models (see Figure 5.3). As described in Section 5.3.1, those samples belong to validation or test set. In addition, in this figure, I indicate the position of the most influential training sample S identified by Infl. As this figure indicates, the sample S is proximal to the samples with negative ground-truth labels for the Twitter dataset (positive for the Fashion dataset). To guarantee the accurate

predictions on those nearby ground-truth samples, it is therefore reasonable to label S as negative (positive for Fashion dataset), which matches the labels provided by Infl but differs from ones given by the human annotators. This indicates the high quality of the labels given by Infl. Thus, when high-quality human labelers are not available, Infl can be an alternative labeler for reducing the labeling overhead without harming the labeling quality.

It is also worth noting that when γ is one where all the training samples are equally weighted. DUTI performs worse than Infl. Based on our observations in the experiments, this phenomenon might be due to the difficulty in *exactly* solving the bi-optimization problem in DUTI, thus producing sub-optimal selections of the influential training samples.

Plus, when $\gamma = 1$, I also observe that Infl-Y performs worse than Infl. Recall that by comparing against Infl, Infl-Y quantifies the influence of each training sample without taking the magnitude of the label changes into the considerations. Since Infl-Y fails to outperform Infl, it thus justifies the necessity of explicitly considering the label changes in the influence function.

On the other hand, when $\gamma = 0$, except MIMIC and Retina, Infl can still beat other baseline methods, thus indicating that the potential of Infl when the uncleaned labels are not included in the training process. Note that for MIMIC and Retina dataset, the performance of Infl is not ideal. One possible reason is that with $\gamma = 0$, the samples with probabilistic labels are not included in the training process, meaning that only a small portion of samples (up to 100) are used for model training. Note that there are 100 samples cleaned in total, thus violating the small cleaning budge assumption. Plus, note that for the influence function method, due to the Taylor expansion in Equation (5.9), one implicit assumption is thus the slight modification on model parameter after small amount of training samples are modified. However, I also observe that significant updates on the model parameters occur after the 100 samples are cleaned for MIMIC and Retina dataset (due to the violation of the small cleaning budge assumption), thus leading to inaccurate estimate on the training sample influence. How to handle this pathological scenario will be also part of our future work.

It is also worth noting that by comparing Table 5.7-5.10 and Table 5.11-5.14, it is worth

noting that with $\gamma = 1$, the model performance is worse with respect to that with $\gamma = 0$, thus implying the negative effect of the probabilistic labels. But when $\gamma = 1$, the strong negative effect of the probabilistic labels do not hurt the performance of Infl, thus suggesting the robustness of Infl when the probabilistic labels are not ideal.

Table 5.3-5.14 also exhibits the benefit of using smaller batch sizes b since it results in better model performance when Infl, especially INFL (two), is used for some datasets (e.g, see its model performance comparison between $b = 100$ and $b = 10$ for Twitter dataset in Table 5.3 and Table 5.5). Intuitively, Infl only quantifies the influence of cleaning *single* training sample rather than multiple ones. Therefore, the larger b is, the more likely that Infl selects a sub-optimal set of b samples for cleaning. Ideally, b should be one, meaning that one training sample is cleaned at each round. However, this can inevitably increase the number of rounds and thus the overall overhead. In **Exp4**, I empirically explored how to choose an appropriate b to balance the model performance and the total running time.

Exp2 The experimental results of **Exp2** are included in Table 5.15-5.16. According to these two tables, Infl still results in much better models than other baseline methods, including TARS. This thus demonstrates the performance advantage of Infl even when the uncleaned labels are all deterministic. So in comparison to TARS, Infl is not only suitable for more general scenarios, but also capable of producing higher-quality models in those scenarios.

Exp3 I present the results of **Exp3** in Table 5.17. As this table indicates, INFL (two) can still achieve the best model performance for those four datasets, thus indicating the potential of applying Infl even when a neural network model is used. Note that the LeNet model is less complicated than other large neural network models, such as ResNet50. Therefore, in the future I would like to do more extensive experiments to evaluate the performance of Infl when large neural network models are used. Since Increm-INFL and DeltaGrad-L are only applicable for strongly convex models such as logistic regression models, I would also like to study how to extend them to handle neural network models.

Exp4 The experimental results are provided in Table 5.20. As this table shows, when

Table 5.20 – Comparison of the model prediction performance (F1 score) with varied b on Twitter dataset (INFL (two))

	uncleaned	b=1000	b=500	b=200	b=100	b=50	b=20	b=10
Twitter	0.6509	0.8672	0.8932	0.8939	0.9149	0.9105	0.9046	0.9064
Fashion	0.6605	0.6942	0.6993	0.6990	0.7042	0.7065	0.7089	0.7082

the cleaning budget is 1000 and b is 100, i.e. roughly 10% of the cleaning budget, the model performance is close to the peak performance. When b becomes smaller, the model performance does not significantly improve and the overall running time increases. Therefore, to balance between model performance and running time, setting b as the 10% of the cleaning budget is recommended.

Exp5 In this experiment, I compare the running time of Increm-INFL and Full in selecting the Top-10 influential training samples (Time_{inf}) at each cleaning round of the loop (2) (with $b = 10$). Due to space, I only include results for the last round in Table 5.19, which are similar to results in other rounds. As Table 5.19 indicates, Increm-INFL is up to 54.7x faster than Full, which is due to the significantly decreased overhead of computing the class-wise gradients for each sample (i.e. Time_{grad}) when Increm-INFL is used. To further illustrate this point, I also record the number of candidate influential training samples whose influence values are explicitly evaluated with and without using Increm-INFL. The result indicates that due to the early removal of uninfluential training samples using Increm-INFL, I only need to evaluate the influence of a small portion of training samples, thus reducing Time_{grad} by up to two orders of magnitude and thereby significantly reducing the total running time, Time_{inf}. In addition, I observe that Increm-INFL always returns the same set of influential training samples as Full, which thus guarantees the correctness of Increm-INFL.

Exp6 Experimental results of **Exp6** are shown in Figure 5.2. The first observation is that DeltaGrad-L can achieve up to 7.5x speed-up with respect to Retrain on updating the model parameters. As shown in Section 5.3.2, the models updated by DeltaGrad-L are not exactly the same as those produced by Retrain, which might cause different model performance between the two methods. However, I observe that the models constructed by those two methods have almost equivalent prediction performance (see the second to last row

in Table 5.5 and Table 5.6). This indicates that it is worthwhile to leverage DeltaGrad-L for speeding up the model constructor.

5.4 Acknowledgement

In this portion of the dissertation, I received extensive help from Dr. James Weimer, who provided the computational resources needed for the experiments, as well as insight on how to design practical solutions in the medical domains.

CHAPTER 6: Extending DeltaGrad and CHEF

Several of the results in this dissertation depend on assumptions on model classes which may not hold in practice, thus preventing their use in more general scenarios. Therefore, this chapter presents ideas for future work on how to extend DeltaGrad and CHEF such that those assumptions could be relaxed. In extending DeltaGrad, I focus on black-box model inversion attacks, which are more realistic than white-box inversion attacks, and build a solution base on the online method (see Section 6.1). To extend CHEF for handling more general models, the main bottleneck is Increm-INFL, for which I leverage an extended L-BFGS algorithm (see Section 6.2.1). I also note that CHEF could be extended in two other ways, including a tight integration between CHEF and weakly supervised learning (see Section 6.2.2), and the combination of CHEF and semi-supervised learning (see Section 6.2.3).

6.1 Extending DeltaGrad

Recall that in Chapter 4, I presented DeltaGrad, an approach to incrementally updating models, which can not only provide significant speed-ups on updating models, but also defend against white-box model inversion attack (see the discussion in Section 1.3). One assumption of this approach is that the models are strongly convex, which, unfortunately, is not true for general neural network models. In fact, neural network models are highly sophisticated and usually over-parameterized, meaning that the number of model parameters is far more than the size of training set. Therefore, such high model complexity brings about huge challenges for designing a solution that can incrementally update the model and simultaneously be resistant to the white-box model inversion attack. Intuitively speaking, the main bottleneck of providing a solution is the need to cache the model parameters and

corresponding gradients at each GD or SGD iteration. This can consume a prohibitively large amount of memory for large over-paremeterized models, but is essential to remove the effect of the deleted training sample throughout the entire training process.

Therefore, in this section, I will focus on a scenario where a *black-box* model inversion attack [101] can occur. This is a more reasonable set-up in practice as mentioned in Section 2.3. Ideally, given a deletion request, to guarantee that an incrementally updated model is as powerful as a model reconstructed from scratch in defending the black-box attack, I expect that the two models can always produce the same output predictions given any input sample such that the adversary is unable to differentiate these two models. As I will explain in Section 6.1.2.2, the online method is resistant to the black-box model inversion attack but vulnerable to the white-box one. Therefore, employing the online method is an appropriate starting point for designing a solution to incrementally update the over-parameterized models and be resistant to *black-box mode inversion attack* at the same time. However, as I will reveal in Section 6.1.2.3, it is insufficient to merely utilize the online method for updating over-parameterized models since it may fail to forget the removed training samples. To remedy this issue, I propose to equip the online method with some additional operations which take the convergence property for the over-parameterized models into account.

The rest of this section begins with an overview of a recent break-through in over-parameterized neural network models, including the description of a property that might be useful for developing the method for incremental updating models. This is followed by a detailed analysis of the online method, including why it fails to defend against white-box model inversion attacks and fails to forget removed training samples from over-parameterized models. This section ends with my proposed solution for incrementally updating over-parameterized models, which leverages both the online method and the one convergence property of over-parameterized models.

6.1.1 Brief introduction to over-parameterized neural network models

It is widely recognized that many advanced neural network models are very powerful in providing accurate predictions, even on unseen data. However, this seems to violate the classical bias-variance trade-off curve, which suggests that neither too simple nor too complicated models are appropriate due to the decreased bias (corresponding to training errors) and the increased variance (corresponding to test errors) with the rise of the model complexity. To reconcile the conflict between the bias-variance trade-off curve and the practice of using neural network models, [166] observed that this curve could be complemented with the double-descent curve. According to this double-descent curve, when the models are under-parameterized, the classical bias-variance trade-off holds while when the model complexity exceeds some threshold—in other words, the models become over-parameterized—the training error approaches zero and the test error gets surprisingly decreased. To fundamentally understand this phenomenon, [167, 168, 169] provide rigorous theoretical analysis from the optimization perspective, which shows that when the number of model parameters in one neural network model is large enough, training this model via gradient descent can provably produce zero training error. In addition, to explain the generalization power of the over-parameterized neural network models, [170] analyzes their generalization error, which is proven to be small enough for large enough training set no matter how complicated the model is.

Apart from the above optimization and generalization properties of the over -parameterized neural network models, researchers also explored other convergence properties for the over-parameterized neural network models. For example, due to the non-convexity of the over-parameterized neural networks, there are more than one local minimum for this type of model, which corresponds to the model parameters producing zero gradient. However, it has been observed that optimizing this type of model via gradient descent converges to a specific local minimum. This phenomenon is referred to as *implicit bias*. For example, as [171] proves, for a linearly separable dataset, there can be infinite number of linear classifiers exactly fitting this dataset, which are also the local minimums of

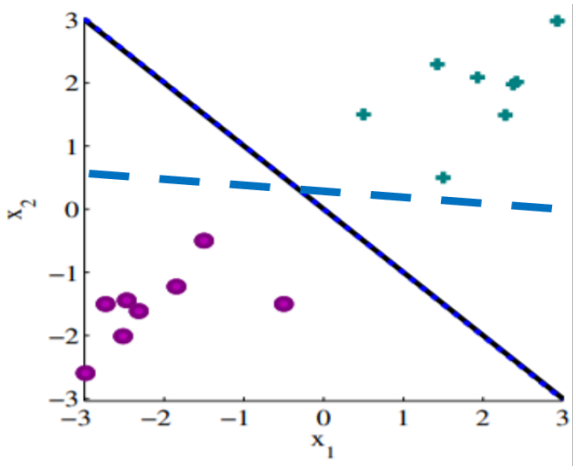


Figure 6.1 – Visual interpretation of the implicit bias when an unregularized logistic regression model is trained on a linearly separable dataset. In this dataset, I use a green '+' and purple dot to denote the positively and negatively labeled training samples, respectively. Note that there can be an infinite number of linear classifiers exactly fitting this dataset, such as the blue solid line and the blue dot line. However, training a logistic regression model via gradient descent method always leads to the blue solid line as the final solution, which is the max-margin solution

the objective function defined by applying logistic regression models. However, training an unregularized logistic regression models on this dataset via gradient descent can provably be biased toward one local minimum, i.e., the linear classifier which produces the maximal margin between the classifier and the training samples (see Figure 6.1 for visual interpretations). For more complicated models such as the over-parameterized two-layer neural network [172], the over-parameterized linear convolutional networks [173] and general over-parameterized neural networks [174], gradient descent method also tends to be biased toward certain local minimum rather than arbitrary local minimums.

6.1.2 Details of the online method

This section gives a detailed description of the online method, which starts by the analysis on the efficiency of the online method on incrementally updating machine learning models, followed by the discussions of why the online method is resistant to the *black-box* model inversion attack but not the *white-box* one. In the end, I analyze why merely utilizing online method fails to accurately update over-parameterized models, which is also empirically

demonstrated with one experiment.

6.1.2.1 Efficiency of the online method

As described in Section 1.3, only a few GD or SGD iterations are required for incrementally updating models by employing online method, which relies on the following appropriate assumptions:

Assumption 6. *The model parameter trained on the full training dataset is* $\boldsymbol{w}^*$

Assumption 7. *The size of the removed training set,* $\mathcal{R}$*, is far smaller than the size of the full training set*

Assumption 8. *With the model parameter* $\boldsymbol{w}^*$*, the gradient evaluated on each training sample* $\boldsymbol{z}$ *is bounded by a constant* G*.*

Note that the above assumptions can be applied for arbitrarily complicated models. Then I derive the following formula based on the above assumptions:

$$\begin{aligned}
&\frac{1}{|\mathcal{Z}_{\text{train}}|}\sum_{\mathbf{z}\in\mathcal{Z}_{\text{train}}}\nabla F(\mathbf{w}^*,\mathbf{z})=0\\
&\Longleftrightarrow \frac{1}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\sum_{\mathbf{z}\in\mathcal{Z}_{\text{train}},\mathbf{z}\notin\mathcal{R}}\nabla F(\mathbf{w}^*,\mathbf{z})=-\frac{1}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\sum_{\mathbf{z}\notin\mathcal{R}}\nabla F(\mathbf{w}^*,\mathbf{z})\\
&\Longleftrightarrow \|\frac{1}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\sum_{\mathbf{z}\in\mathcal{Z}_{\text{train}},\mathbf{z}\notin\mathcal{R}}\nabla F(\mathbf{w}^*,\mathbf{z})\|=\|-\frac{1}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\sum_{\mathbf{z}\notin\mathcal{R}}\nabla F(\mathbf{w}^*,\mathbf{z})\|\\
&\leq \frac{|\mathcal{R}|G}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\approx 0
\end{aligned} \tag{6.1}$$

The second step moves the gradient on the removed training samples to the right-hand side of the equality and the last step leverages Assumption 8. This tells us that with model parameter $\mathbf{w}^*$, the sum of the gradients on the remaining training samples is close to zero, meaning that $\mathbf{w}^*$ is close to one local minimum for the objective function $\frac{1}{|\mathcal{Z}_{\text{train}}-\mathcal{R}|}\sum_{\mathbf{z}\in\mathcal{Z}_{\text{train}},\mathbf{z}\notin\mathcal{R}}F(\mathbf{w}^*,\mathbf{z})$. Therefore, I can easily run a few GD or SGD updates to identify one such local minimum $\mathbf{w}_1^*$.

It is worth noting that for strongly convex models, the online method always produces incrementally updated models that are the same as the ones retrained from scratch on the remaining training samples. This relies on the uniqueness of the local minimum of strongly convex models, which is also the global minimum.

6.1.2.2 *Online method* and the model inversion attack

In this section, I analyze why the online method can defend against the *black-box* model inversion attack but is vulnerable to the *white-box* attack.

As shown in Section 6.1.2, the strongly convex models incrementally updated by the online method are exactly the same as the ones retrained from scratch on the remaining training samples. This indicates that the two models always produce the same output given any input training samples and thus cannot be differentiated by the adversary who only has black-box access to the models. Therefore, one can claim that the online method is able to defend against the *black-box model inversion attack*.

In contrast, by assuming that one training sample is deleted, I can craft the following *white-box* attack on the updated model by the online method: 1) undo the GD or SGD updates for t iterations such that the updated models could be rolled back to the model status before the deletions happen, which can be accomplished by performing t iterations of gradient ascent or stochastic gradient ascent on the remaining training samples (recall that t is the number of GD or SGD iterations that the *online method* requires for incremental updates); 2) estimate the gradient evaluated on the deleted training sample at the current model status by leveraging the fact that the gradient on the full training set is close to zero due to the convergence condition (recall that the adversary may know the remaining training samples after the deletion requests); 3) reconstruct the deleted sample by employing the solution from [175], which only uses the gradient on the deleted training sample. Note that this problem will not arise for PrIU and DeltaGrad as long as the provenance information for incremental updates is not leaked. Intuitively, PrIU and DeltaGrad “erase" the footprint of the deleted training sample from *every* GD or SGD iteration so that the gradient evaluated

on the removed training sample cannot be obtained by the adversary, thus safeguarding the removed training samples.

6.1.2.3 Failure of the *online method* for incrementally updating over-parameterized models

Another limitation of the online method is that it cannot accurately update more complicated models, e.g., the over-parameterized neural network models. Intuitively speaking, due to over-parameterization, different training samples may be memorized by different portions of model parameters. Therefore, after the deletion request, if the *online method* is used without additional operations to explicitly forget the removed training samples from the models, the model parameters for memorizing those samples will not be affected.

To justify this point, I provide some empirical evidence. To start with, I add 5 randomly generated noisy training samples into the training set of cifar10 dataset [176] and then train a VGG16 model [177] on this augmented training set by running SGD for long enough. Note that the VGG16 model is composed of up to 138 million parameters, which is far more than the size of cifar10 dataset (i.e. 60000) and thus could be regarded as an over-parameterized model. After the termination of the training process, it is worth noting that the resulting model could memorize all the training samples, including those noisy samples, due to the correct model predictions on those samples, which is consistent with the observation in [178]. Afterwards, the online method is applied to incrementally update this model by continuing running SGD updates after the noisy training samples are excluded from the training set. Unfortunately, I observed that no matter how the hyper-parameters (e.g. mini-batch size and epochs) of the online method are varied, the predictions on those deleted training samples by using this incrementally updated model are still all correct. In contrast, if the model is updated by retraining from scratch on the original training set of cifar10 dataset (without including the noisy samples), then this model would give pretty random prediction results on those noisy samples. This thus indicates that simply applying the online method cannot clear the memorization of the removed training samples from the over-parameterized

models.

6.1.3 Proposed solution

To facilitate the incremental updates on the over-parameterized models, I proposed to equip the online method with the implicit bias property for over-parameterized models such that the incrementally updated models could match the ones by retraining from scratch.

Specifically, after the online method is applied, it is worth noting that this method would end up with a local minimum of the model constructed on the remaining training samples, which may not necessarily satisfy the implicit bias property. To identify one local minimum satisfying this property, one possible way is to search within the parameter space comprised of all the local minimums rather than the entire parameter space, which can thus be formalized as the following constrained optimization problem:

$$\begin{aligned} &\text{implicit bias property holds for } \mathbf{w} \\ &s.t. \| \frac{1}{|\mathcal{Z}_{\text{train}} - \mathcal{R}|} \sum_{\mathbf{z} \in \mathcal{Z}_{\text{train}}, \mathbf{z} \notin \mathcal{R}} \nabla F(\mathbf{w}, \mathbf{z}) \| = 0 \end{aligned} \tag{6.2}$$

Solving the above formula can take $\mathbf{w}_1^*$ as the initialized value (recall that $\mathbf{w}_1^*$ is the local minimum identified by the online method). I can then instantiate the above optimization problem by explicitly considering the implicit bias property for different specific over-parameterized models. For example, by following the problem set-up in [171], i.e. optimizing a logistic regression model on linearly separable data, Equation (6.2) could be instantiated as:

$$\min_{\mathbf{w}} \|\mathbf{w}\|, s.t. \| \frac{1}{|\mathcal{Z}_{\text{train}} - \mathcal{R}|} \sum_{\mathbf{z} \in \mathcal{Z}_{\text{train}}, \mathbf{z} \notin \mathcal{R}} \nabla F(\mathbf{w}, \mathbf{z}) \| = 0 \tag{6.3}$$

In the future, I would like to explore how to instantiate Equation (6.2) for general neural network models, which may utilize the implicit bias property discovered by [179].

6.2 Extending CHEF

This section considers how to extend CHEF to handle more complicated scenarios, which starts by discussing how of extending CHEF to deal with general machine learning models, followed by some initial thoughts on how to combine CHEF and weakly supervised learning as well as semi-supervised learning.

6.2.1 Extending CHEF for general machine learning models

Recall that in Chapter 5, I assume that the machine learning models are strongly convex for Increm-INFL and DeltaGrad-L. Note that DeltaGrad-L is a variant of DeltaGrad. Therefore, as long as DeltaGrad is capable of incrementally updating general neural networks by following the ideas in Section 6.1, it is not difficult to adjust DeltaGrad-L to support general models. Therefore, in this section, I primarily focus on how to generalize DeltaGrad-L to deal with more complicated models.

Recall that as the first step toward deriving the perturbation bound on $\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma)$, the formula $-\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma) - \mathcal{I}_0(\tilde{\mathbf{z}}, \delta_y, \gamma)$ is partitioned into two components, i.e. Diff_1 and Diff_2 (see Equation (5.12) in Section 5.2.1.3). I note that Diff_1 could be rewritten into Equation (5.13), which requires to compute a Hessian-vector product, $\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})$. In the proof of Theorem 13, Equation (5.13) is bounded by utilizing the L2 norm of $\mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})$, which depends on the strong convexity assumption on the model class and might over-estimate the bound on Diff_1.

However, it is worth noting that the Hessian-vector product, $\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})$ in the term Diff_1 could be computed in an alternative way, e.g. by leveraging the L-BFGS algorithm [98, 29, 32, 99, 31, 100, 30]. I note that L-BFGS algorithm can also be extended to the case where the models are not necessarily convex (see e.g. [47]). As a result, if the Hessian-vector product, $\mathbf{v}^\top \mathbf{H}^{(j)}(\mathbf{w}^{(k)}, \mathbf{z})$, could be estimated with the extended L-BFGS algorithm for non-convex models, then the derived bound on $\mathcal{I}^{(k)}_{\text{pert}}(\tilde{\mathbf{z}}, \delta_y, \gamma)$ could be thus applied for non-convex models, thus facilitating the use of Increm-INFL in more general scenarios.

6.2.2 A tight integration between CHEF and weakly supervised learning

As described in Section 5.3.1.3, if the probabilistic labels are produced by following the solutions in [48], [159] or [160], then those probabilistic labels depend on small number of labeled training samples. In CHEF, after the labels of selected training samples are cleaned at each cleaning iteration, one can use the augmented set of labeled training samples to further update the probabilistic labels for the remaining uncleaned training samples. This can improved the overall label quality, but is not considered in CHEF. Therefore, in this section, I discuss how to address this by tightly integrating CHEF with weakly supervised learning. The main bottleneck is adjusting INFL to take into account the updates of the probabilistic labels on the uncleaned training samples.

To address this problem, recall that for INFL in Chapter 5, to estimate the influence of cleaning the label of each uncleaned training sample, Equation (5.8) is analyzed first, which, is modified as follows when the updates of probabilistic labels occur due to the label cleaning operations on the sample $\tilde{\mathbf{z}}_r$:

$$\begin{aligned} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}(\mathbf{w}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} F(\mathbf{w},\mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma F(\mathbf{w},\tilde{\mathbf{z}}_i)] \\ + \epsilon_1 F(\mathbf{w},\tilde{\mathbf{z}}_r(\delta_{y,r})) - \epsilon_2 F(\mathbf{w},\tilde{\mathbf{z}}_r) \\ - \epsilon_3 \sum\nolimits_{i\neq r} F(\mathbf{w},\tilde{\mathbf{z}}_i) + \epsilon_3 \sum\nolimits_{i\neq r} F(\mathbf{w},\tilde{\mathbf{z}}_i(\delta_{y,i}(\delta_{y,r}))) \end{aligned} \tag{6.4}$$

in which, $\delta_{y,r}$ denotes the label updates on the sample $\tilde{\mathbf{z}}_r$ after this sample is cleaned and $\delta_{y,i}(\delta_{y,r}), i \neq r$ denotes the label updates on other uncleaned training samples triggered by the label updates on the sample $\tilde{\mathbf{z}}_r$. Then by denoting the minimum of the above objective function as $\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}$ and following the same derivation in the proof of Equation (5.4), i.e.,

Section 5.2.1.2, the following formula could be derived:

$$\begin{aligned}\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}} - \hat{\mathbf{w}} &= -\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\epsilon_1\nabla_{\mathbf{w}}F\left(\hat{\mathbf{w}},\tilde{\mathbf{z}}_r(\delta_{y,r})\right) - \epsilon_2\nabla_{\mathbf{w}}F\left(\hat{\mathbf{w}},\tilde{\mathbf{z}}_r\right)\right.\\ &\quad \left. - \epsilon_3\sum\nolimits_{i\neq r}\nabla_{\mathbf{w}}F\left(\mathbf{w},\tilde{\mathbf{z}}_i\right) + \epsilon_3\sum\nolimits_{i\neq r}\nabla_{\mathbf{w}}F\left(\mathbf{w},\tilde{\mathbf{z}}_i(\delta_{y,i}(\delta_{y,r}))\right)\right]\\ &= -\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{z}}}\left(\hat{\mathbf{w}}\right)^{-1}\left[\epsilon_1\nabla_{\mathbf{w}}F\left(\hat{\mathbf{w}},\tilde{\mathbf{z}}_r(\delta_{y,r})\right) - \epsilon_2\nabla_{\mathbf{w}}F\left(\hat{\mathbf{w}},\tilde{\mathbf{z}}_r\right)\right.\\ &\quad \left. + \epsilon_3\sum\nolimits_{i\neq r}\nabla_{\mathbf{w}}\nabla_y F\left(\mathbf{w},\tilde{\mathbf{z}}_i\right)\delta_{y,i}(\delta_{y,r})\right]\end{aligned} \tag{6.5}$$

The last step utilizes the Cauchy mean value theorem and the fact that $\nabla_{\mathbf{w}}F\left(\mathbf{w},\tilde{\mathbf{z}}_i\right)$ is linear in the label of $\tilde{\mathbf{z}}_i$. Therefore, as long as the quantity $\delta_{y,i}(\delta_{y,r})$ in the above formula is obtained, one can thus obtain the adjusted influence function when the probabilistic labels are also dynamically updated after the label of the sample $\tilde{\mathbf{z}}_r$ is cleaned. But note that the updated probabilistic labels (and thus $\delta_{y,i}(\delta_{y,r})$) are obtained after the solution in [48], [159] or [160] is re-applied on the updated set of cleaned training samples, which requires to train complicated generative models. Therefore, it is crucial to estimate the quantity $\delta_{y,i}(\delta_{y,r})$ efficiently without reconstructing those generative models from scratch since $\delta_{y,i}(\delta_{y,r})$ varies with varied training sample, $\mathbf{z}_r$, and the updates on its label, $\delta_{y,r}$, which will be left as future work.

6.2.3 Integrating CHEF with semi-supervised learning

As described in Section 2.4, when a plethora of training samples are unlabeled, there is a trend in combining active learning with semi-supervised learning for developing better models and saving the human labeling cost at the same time. Since the method INFL could be regarded as an alternative to the classical active learning methods for selecting unlabeled samples for labeling, it is thus worth considering its integration with semi-supervised learning, which could follow the framework proposed by [56].

In [56], the machine learning model is constructed by optimizing the following semi-

supervised learning (SSL) based objective function:

$$F(\mathbf{w}) = \frac{1}{N}[\sum_{i=1}^{N_d} F(\mathbf{w}, \mathbf{z}_i) + \gamma \sum_{i=1}^{N_p} F^{SSL}(\mathbf{w}, \tilde{\mathbf{x}}_i)] \tag{6.6}$$

Here, I borrow notation used in Equation (5.1) from Chapter 5. For example, $F(\mathbf{w}, \mathbf{z}_i)$ still denotes the loss on a labeled training sample $\mathbf{z}_i = (\mathbf{x}_i, y_i)$. Note that for all the other training samples except the labeled training sample set $\{\mathbf{z}_i\}_{i=1}^{N_d}$, Equation (5.1) calculates the loss on those samples by utilizing their probabilistic labels. In contrast, those samples are assumed to be unlabeled in the loss, $F^{SSL}(\mathbf{w}, \tilde{\mathbf{z}}_i)$ in Equation (6.6), which is the loss developed in semi-supervised learning for unlabeled training samples. There are a variety of choices for this loss. In [56], the following loss is used to minimize the sensitivity with respect to the small perturbations on each unlabeled training sample:

$$\begin{aligned} &F^{SSL}(\mathbf{w}, \tilde{\mathbf{x}}_i) = D([p^{(0)}(\mathbf{w}, \tilde{\mathbf{x}}_i), p^{(1)}(\mathbf{w}, \tilde{\mathbf{x}}_i), \ldots, p^{(C-1)}(\mathbf{w}, \tilde{\mathbf{x}}_i)] \\ &, [p^{(0)}(\mathbf{w}, \tilde{\mathbf{x}}_i'), p^{(1)}(\mathbf{w}, \tilde{\mathbf{x}}_i'), \ldots, p^{(C-1)}(\mathbf{w}, \tilde{\mathbf{x}}_i')]) \end{aligned} \tag{6.7}$$

I reuse the notation, $p^{(j)}(\mathbf{w}, \tilde{\mathbf{x}}_i), j = 0, 1, \ldots, C-1$, from Equation (5.6) to denote the probability that $\tilde{\mathbf{x}}_i$ is predicted as the class j by the model. In addition, $\tilde{\mathbf{x}}_i'$ represents the sample feature after a small and model-free perturbation is added to $\tilde{\mathbf{x}}_i$ and the function $D(\cdot, \cdot)$ is used to measure the model output difference between the $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_i'$, which could be the KL-divergence measure.

Then given the objective function in Equation (6.6), I can then follow the same intuition as INFL in Chapter 5 to derive an influence function such that it can reflect how labeling a certain unlabeled training sample influences the prediction performance, i.e.:

$$\begin{aligned} &\mathcal{I}_{\text{ssl}}(\tilde{\mathbf{x}}, y, \gamma) \approx N \cdot (F(\mathbf{w}^U, \mathcal{Z}_{\text{val}}) - F(\mathbf{w}, \mathcal{Z}_{\text{val}})) \\ &= -\nabla_{\mathbf{w}} F(\mathbf{w}, \mathcal{Z}_{\text{val}})^\top \mathbf{H}^{-1}(\mathbf{w})[\nabla_{\mathbf{w}} F(\mathbf{w}, (\tilde{\mathbf{x}}, y)) - \gamma \nabla_{\mathbf{w}} F^{SSL}(\mathbf{w}, \tilde{\mathbf{x}})], \end{aligned} \tag{6.8}$$

Here y represents some possible label for the sample $\tilde{\mathbf{x}}$ and $\mathbf{H}(\mathbf{w})$ represents the Hessian matrix evaluated on the objective function, Equation (6.6). Then Equation (6.8) could be

employed for selecting the most informative samples for labeling and providing possibly cleaned labels for those samples. The derivation of Equation (6.8) is provided below.

Proof. Similar to the derivation of Equation (5.4), i.e. Section 5.2.1.2, the following objective function is considered first for an uncleaned training sample $\tilde{\mathbf{x}}$ and one of its possibly clean label y:

$$\begin{aligned} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\mathbf{w}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} F(\mathbf{w},\mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma F^{SSL}(\mathbf{w},\tilde{\mathbf{x}}_i)] \\ + \epsilon_1 F(\mathbf{w},(\tilde{\mathbf{x}},y)) - \epsilon_2 F^{SSL}(\mathbf{w},\tilde{\mathbf{x}}_i) \end{aligned} \tag{6.9}$$

By denoting the minimizer of the formula above as $\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}$, the following equation on $\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}$ holds:

$$\begin{aligned} \nabla_{\mathbf{w}} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}) = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},\mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F^{SSL}(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},\tilde{\mathbf{x}}_i)] \\ + \epsilon_1 \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},(\tilde{\mathbf{x}},y)) - \epsilon_2 \nabla_{\mathbf{w}} F^{SSL}(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},\tilde{\mathbf{x}}) = 0 \end{aligned}$$

Similar to the derivation of Equation (5.4), I again represent the minimizer of $F_{0,0,\tilde{\mathbf{x}},y}(\mathbf{w})$ as $\hat{\mathbf{w}}$, which is also the minimizer of Equation (6.7). Then due to the closeness of $\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}$ and $\hat{\mathbf{w}}$ as both ϵ_1 and ϵ_2 are near-zero values, I can then apply Taylor expansion on $\nabla_{\mathbf{w}} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},\epsilon_1,\epsilon_2)$, i.e.:

$$\begin{aligned} 0 = \nabla_{\mathbf{w}} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y},\epsilon_1,\epsilon_2) \approx \nabla_{\mathbf{w}} F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}},\epsilon_1,\epsilon_2) + \mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}})(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y} - \hat{\mathbf{w}}) \\ = \frac{1}{N}[\sum\nolimits_{i=1}^{N_d} \nabla_{\mathbf{w}} F(\hat{\mathbf{w}},\mathbf{z}_i) + \sum\nolimits_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F(\hat{\mathbf{w}},\tilde{\mathbf{x}}_i)] \\ + \epsilon_1 \nabla_{\mathbf{w}} F(\hat{\mathbf{w}},(\tilde{\mathbf{x}},y)) - \epsilon_2 \nabla_{\mathbf{w}} F^{SSL}(\hat{\mathbf{w}},\tilde{\mathbf{x}}) + \mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}})(\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y} - \hat{\mathbf{w}}), \end{aligned} \tag{6.10}$$

in which $\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(*)$ denotes the Hessian matrix of $F_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\mathbf{w})$. Then by using the fact that $\frac{1}{N}[\sum_{i=1}^{N_d} \nabla_{\mathbf{w}} F(\hat{\mathbf{w}},\mathbf{z}_i) + \sum_{i=1}^{N_p} \gamma \nabla_{\mathbf{w}} F(\hat{\mathbf{w}},\tilde{\mathbf{x}}_i)] = 0$ (since $\hat{\mathbf{w}}$ is the minimizer of $F_{0,0,\tilde{\mathbf{x}},y}(\mathbf{w})$) and $\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}}) \approx \mathbf{H}_{0,0,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}}) = \mathbf{H}(\hat{\mathbf{w}})$ (since ϵ_1 and ϵ_2 are near zero, recall

that $\mathbf{H}(*)$ is the Hessian matrix of Equation (5.1)), the formula above is derived as:

$$\hat{\mathbf{w}}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y} - \hat{\mathbf{w}} = -\mathbf{H}_{\epsilon_1,\epsilon_2,\tilde{\mathbf{x}},y}(\hat{\mathbf{w}})^{-1}\left[\epsilon_1 \nabla_{\mathbf{w}} F(\hat{\mathbf{w}}, (\tilde{\mathbf{x}}, y)) - \epsilon_2 \nabla_{\mathbf{w}} F^{SSL}(\hat{\mathbf{w}}, \tilde{\mathbf{x}})\right]$$

Considering the similarity between the above formula and Equation (5.10) in Section 5.2.1.2, the rest of the derivation would be same as that in Section 5.2.1.2, which would eventually result in Equation (6.8). □

Discussion Similar to INFL in CHEF, Equation (6.8) could be evaluated for every training sample with all possible deterministic labels. Therefore, this could not only determine which unlabeled training samples should be labeled by the human annotators, but also automatically suggest possibly clean labels. As a consequence, due to the lower annotation overhead, one can expect that this method would be preferable if it does not perform significantly worse than the modified active learning method in [56]. This empirical comparison is left as future work.

CHAPTER 7: Conclusions

I conclude the dissertation by summarizing its contributions, and presenting a vision for future work.

7.1 Summary

My dissertation concerns leveraging *fine-grained provenance* to deal with emerging data science applications, including the problem of producing fine-grained data citations, the problem of incrementally updating machine learning models upon the deletion of training samples, and the problem of reducing the overhead in the pipeline of cleaning label uncertainties.

To produce fine-grained data citations, I provided two types of approaches, i.e. the Rewriting-based approach and the Provenance-based approach. The former utilizes fine-grained provenance implicitly for producing citations for simple queries and views, i.e. conjunctive queries and conjunctive views, while the latter utilizes fine-grained provenance explicitly to handle a more general type of queries and views, i.e. aggregate queries and aggregate views. Note that leveraging fine-grained provenance for large datasets can incur extremely high overhead, which motivates a series of optimization techniques in both the Rewriting-based approach and the Provenance-based approach for speed-ups.

To facilitate incremental updates on machine learning models after small number of training samples are removed, I proposed two solutions, i.e. PrIU and DeltaGrad. The former method aims at incrementally updating linear regression, logistic regression models and possibly generative additive models, which is accomplished by linearizing the non-linear components in their GD or SGD update rules, followed by explicitly utilizing the provenance-semiring framework extended to linear algebra formulas. In contrast to PrIU, DeltaGrad

is capable of incrementally updating a more general class of models, i.e., strongly convex models, by using pre-cached information (which can be regarded as provenance information) and the L-BFGS algorithm to incrementally maintain the gradients calculated at each GD or SGD iteration. Extensive experiments on benchmark datasets demonstrate that both approaches achieve significant speed-ups on updating ML models in comparison to reconstructing the models from scratch. I also proposed some initial ideas on how to incrementally maintain general machine learning models as part of the future work.

Based on this solution to incrementally update ML models, I then developed CHEF to reduce the overhead and cost at each phase of the label cleaning pipeline. This solution consists of 1) INFL to identify the most influential training samples for cleaning and automatically suggesting cleaned labels; 2) Increm-INFL to reduce the overhead of evaluating training sample influence with INFL by filtering out uninfluential training samples early; 3) DeltaGrad-L (adapted from DeltaGrad) to incrementally update ML models after the labels of the most influential training are cleaned; 4) a redesign of the label cleaning pipeline to allow human annotators to clean smaller batch of training samples, which enables better overall model performance and potentially early termination of the label cleaning process once satisfactory model performance is reached. I also discussed how to extend CHEF to handle more general machine learning models, and how to integrate CHEF with weakly-supervised and semi-supervision learning for further model performance improvements.

7.2 Future work

In the near future, it would be interesting to see how my solutions for incrementally updating machine learning models can be used in various real applications. For example, in the state-of-the-art provenance-enabled data science platforms, the input-output dependency is constructed for the entire pipeline such that the results could be refreshed effectively with provenance after the updates of the input or parameters (see e.g., [180, 181, 182]). But the machine learning models appearing in those platforms can only be updated from scratch to reflect the updates of input data. Therefore, I can envision that as the solutions for

incrementally updating machine learning models become more mature, it would be beneficial to integrate these solutions into the above data science tools such that the provenance support in those tools become more efficient. Similar situations occur in the application of optimizing DBMS internals with neural network models. Typical examples include learning-based indexes [183], learning-based join order selections [184] and learning-based cardinality estimation [185]. However, under OLTP-like workloads, especially when certain records are deleted from the databases, it still remains a challenge on how to effectively refresh the learned models by utilizing the above techniques. Hence, it would be an inspiring option to integrate the solution to incrementally update machine learning models with the above learning-based techniques inside DBMSs such that those techniques could be compatible to realistic database workloads.

We also note that CHEF is only a first step towards a full-fledged solution for label cleaning in the medical domain. This is because there are considerable practical concerns that still need to be addressed. For example, confidence intervals of prediction results and cleaned labels are important for physician to understand how certain the results are, which is crucial for making life-critical decisions [186]. It is also worth noting that privacy is a big concern in the medical domain, meaning that the medical records might be encrypted before the model training phase [187]. Therefore, modifications to CHEF are required to make it suitable for such privacy-preserving scenarios. Focusing on these practical considerations is an interesting future research direction to explore.

In addition to data provenance, other database techniques could be used for constructing high-quality models. For example, in addition to efficiently refining the labels of training samples, which is what CHEF addresses, other data preparation steps are essential to guarantee the quality of the training data. These operations include integrating data from various sources [188], extracting structured data from text corpora [189], crowdsourcing data for model training [190], and cleaning training sample features [2], which all require database-related techniques. In particular, incremental update techniques for these operations have not been well-studied yet, but are important in practice when the underlying raw

data changes. Therefore, as another part of the future work, I will explore whether the ideas of incremental computation in CHEF can be applied to these more general data preparation operations.

Beyond the need for incremental computation in the data science pipeline, I am also interested in interpretability of complex models such as neural network models. It is widely known that although neural network models are very powerful in providing accurate predictions in various classification tasks, these models are generally regarded as black boxes and thus it is challenging to interpret why certain predictions are given. However, such interpretability is essential in many safety-critical applications, such as self-driving cars, portfolio management in financial domain and disease diagnostics in medical domain [20]. In fact, different types of interpretations could be given for the model predictions, such as interpreting "which feature of the input (test) sample is most important for the model output" and "which training sample is most important for the model output of a certain input (test) sample". These two questions could be answered by the feature-level Shapley value [191] and the instance-level Shapley value (i.e. Data Shapley value) respectively [5]. As discussed in Section 1.3, evaluating the Data Shapley value needs to refresh the models repetitively after the small updates on the training set. Therefore, the solutions to incrementally update machine learning models could support efficient evaluations of the Data Shapley value. In the future, I would love to explore whether data provenance could benefit other types of explanations for the model predictions, such as explanations of feature importance by utilizing the feature-level Shapley value, which may require the use of fine-grained provenance at the feature level [25]. Other than the efficiency issues mentioned above, there also remain other issues in the model interpretability problem. For example, explaining the entire model behaviors (referred to as *global interpretation*) is significantly more challenging than providing explanations for a single prediction (referred to as *local interpretation*) [192]. Hence, I would also love to explore how to deal with this global interpretation problem and other related issues on model interpretability in future.

Last but not least, due to the strong connections between data provenance and logical

reasoning, I would also like to explore the use of logical reasoning with neural network models, which is another emerging research topic in machine learning community. It appears that logic can be combined with neural network models in various ways to take advantage of the prediction power of the neural network models and the interpretability of logical reasoning at the same time. For example, [193] proposes to evaluate certain predicates in the first-order logic rules with the model output, which could leverage the background knowledge encoded by the logic rules to enhance the model prediction performance. However, the integration between the neural network models and logic could also bring about new challenges. To name a few, how to optimize a neural network model to fit a training dataset has been extensively studied in the last few decades. However, it is still unclear how to perfectly optimize neural network models when the logic rules are involved in the training process. It also still remains a difficult task to effectively extract knowledge from neural network models [194].

BIBLIOGRAPHY

[1] X. Niu, B. Glavic, D. Gawlick, Z. H. Liu, V. Krishnaswamy and V. Radhakrishnan, "Interoperability for provenance-aware databases using prov and json," in *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[2] S. Krishnan, J. Wang, E. Wu, M. J. Franklin and K. Goldberg, "Activeclean: interactive data cleaning for statistical modeling," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 948–959, 2016.

[3] D. Kang, D. Raghavan, P. Bailis and M. Zaharia, *Model assertions for debugging machine learning*, Preprint, 2018.

[4] A. Heidari, J. McGrath, I. F. Ilyas and T. Rekatsinas, "Holodetect: few-shot learning for error detection," *arXiv preprint arXiv:1904.02285*, 2019.

[5] A. Ghorbani and J. Zou, "Data shapley: equitable valuation of data for machine learning," in *International Conference on Machine Learning*, 2019, pp. 2242–2251.

[6] S. B. Davidson, D. Deutsch, T. Milo and G. Silvello, "A model for fine-grained data citation," in *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Online Proceedings*, 2017.

[7] T. J. Green, G. Karvounarakis and V. Tannen, "Provenance semirings," in *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 2007, pp. 31–40.

[8] Y. Amsterdamer, D. Deutch and V. Tannen, "Provenance for aggregate queries," in *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 2011, pp. 153–164.

[9] P. Buneman, S. Khanna and T. Wang-Chiew, "Why and where: a characterization of data provenance," in *International conference on database theory*, Springer, 2001, pp. 316–330.

[10] T. J. Green, G. Karvounarakis, Z. G. Ives and V. Tannen, "Update exchange with mappings and provenance," in *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, pp. 675–686.

[11] S. B. Davidson, D. Deutch, T. Milo and G. Silvello, "A model for fine-grained data citation.," in *CIDR*, 2017.

[12] Y. Wu, A. Alawini, S. B. Davidson and G. Silvello, "Data citation: giving credit where credit is due," in *Proceedings of the 2018 International Conference on Management of Data*, ACM, 2018, pp. 99–114.

[13] Y. Wu, A. Alawini, D. Deutch, T. Milo and S. Davidson, "Provcite: provenance-based data citation," *Proceedings of the VLDB Endowment*, vol. 12, no. 7, pp. 738–751, 2019.

[14] A. J. Pawson, J. L. Sharman, H. E. Benson, E. Faccenda, S. P. Alexander, O. P. Buneman, A. P. Davenport, J. C. McGrath, J. A. Peters, C. Southan *et al.*, "The iuphar/bps guide to pharmacology: an expert-driven knowledgebase of drug targets and their ligands," *Nucleic acids research*, vol. 42, no. D1, pp. D1098–D1106, 2014.

[15] J. F. Armstrong, E. Faccenda, S. D. Harding, A. J. Pawson, C. Southan, J. L. Sharman, B. Campo, D. R. Cavanagh, S. P. Alexander, A. P. Davenport *et al.*, "The iuphar/bps guide to pharmacology in 2020: extending immunopharmacology content and introducing the iuphar/mmv guide to malaria pharmacology," *Nucleic acids research*, vol. 48, no. D1, pp. D1006–D1021, 2020.

[16] S. Proell and A. Rauber, "A scalable framework for dynamic data citation of arbitrary structured data.," in *DATA*, 2014, pp. 223–230.

[17] L. B. Honor, C. Haselgrove, J. A. Frazier and D. N. Kennedy, "Data citation in neuroimaging: proposed best practices for data identification and attribution," *Frontiers in neuroinformatics*, vol. 10, p. 34, 2016.

[18] S. Rizvi, A. Mendelzon, S. Sudarshan and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 551–562.

[19] F. Psallidas and E. Wu, "Smoke: fine-grained lineage at interactive speed," *Proceedings of the VLDB Endowment*, vol. 11, no. 6, pp. 719–732, 2018.

[20] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, p. 93, 2018.

[21] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[22] M. H. Quenouille, "Notes on bias in estimation," *Biometrika*, vol. 43, no. 3/4, pp. 353–360, 1956.

[23] M. Fredrikson, S. Jha and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[24] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich and V. Tannen, "Putting lipstick on pig: enabling database-style workflow provenance," *Proceedings of the VLDB Endowment*, vol. 5, no. 4, pp. 346–357, 2011.

[25] Z. Yan, V. Tannen and Z. G. Ives, "Fine-grained provenance for linear algebra operators," in *Proceedings of the 8th USENIX Conference on Theory and Practice of Provenance*, USENIX Association, 2016, pp. 1–6.

[26] R. Kress, "Interpolation," in *Numerical Analysis*. New York, NY: Springer New York, 1998, pp. 151–188, ISBN: 978-1-4612-0599-9.

[27] Y. Wu, V. Tannen and S. B. Davidson, "Priu: a provenance-based approach for incrementally updating regression models," *arXiv preprint arXiv:2002.11791*, 2020.

[28] T. Hastie and R. Tibshirani, "Generalized additive models," *Statistical Science*, vol. 1, no. 3, pp. 297–318, 1986.

[29] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Mathematics of computation*, vol. 35, no. 151, pp. 773–782, 1980.

[30] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory bfgs," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3151–3181, 2015.

[31] C. Zhu, R. H. Byrd, P. Lu and J. Nocedal, "Algorithm 778: l-bfgs-b: fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[32] R. H. Byrd, J. Nocedal and R. B. Schnabel, "Representations of quasi-newton matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 1-3, pp. 129–156, 1994.

[33] S. Neel, A. Roth and S. Sharifi-Malvajerdi, "Descent-to-delete: gradient-based methods for machine unlearning," in *Algorithmic Learning Theory*, PMLR, 2021, pp. 931–962.

[34] T. Olatunji, L. Yao, B. Covington, A. Rhodes and A. Upton, "Caveats in generating medical imaging labels from radiology reports," *arXiv preprint arXiv:1905.02283*, 2019.

[35] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu and C. RÃł, "Snorkel: rapid training data creation with weak supervision," in *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, NIH Public Access, vol. 11, 2017, p. 269.

[36] L Smyth, "Training-valuenet: a new approach for label cleaning on weakly-supervised datasets," 2020.

[37] S. H. Bach, D. Rodriguez, Y. Liu, C. Luo, H. Shao, C. Xia, S. Sen, A. Ratner, B. Hancock, H. Alborzi *et al.*, "Snorkel drybell: a case study in deploying weak supervision at industrial scale," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 362–375.

[38] M. Nashaat, A. Ghosh, J. Miller and S. Quader, "Wesal: applying active supervision to find high-quality labels at industrial scale," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[39] D. Hao, L. Zhang, J. Sumkin, A. Mohamed and S. Wu, "Inaccurate labels in weakly-supervised deep learning: automatic identification and correction and their impact on classification performance," *IEEE journal of biomedical and health informatics*, vol. 24, no. 9, pp. 2701–2710, 2020.

[40] M. Mahdavi, F. Neutatz, L. Visengeriyeva and Z. Abedjan, "Towards automated data cleaning workflows," *Machine Learning*, vol. 15, p. 16, 2019.

[41] J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, B. Haghgoo, R. Ball, K. Shpanskaya *et al.*, "Chexpert: a large chest radiograph dataset with uncertainty labels and expert comparison," in *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[42] Y. Wu, J. Weimer and S. B. Davidson, "Chef: a cheap and fast pipeline for iteratively cleaning label uncertainties," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, 2021.

[43] Y. Wu, J. Weimer and S. B. Davidson, "Chef: a cheap and fast pipeline for iteratively cleaning label uncertainties (technical report)," *arXiv preprint arXiv:2107.08588*, 2021.

[44] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," *arXiv preprint arXiv:1703.04730*, 2017.

[45] B. Settles, "Active learning literature survey," 2009.

[46] J. Huang, L. Qu, R. Jia and B. Zhao, "O2u-net: a simple noisy label detection approach for deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3326–3334.

[47] X. Wang, S. Ma, D. Goldfarb and W. Liu, "Stochastic quasi-newton methods for nonconvex stochastic optimization," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 927–956, 2017.

[48] B. Boecking, W. Neiswanger, E. Xing and A. Dubrawski, "Interactive weak supervision: learning useful heuristics for data labeling," *arXiv preprint arXiv:2012.06046*, 2020.

[49] S. Biegel, R. El-Khatib, L. O. V. B. Oliveira, M. Baak and N. Aben, "Active weasul: improving weak supervision with active learning," *arXiv preprint arXiv:2104.14847*, 2021.

[50] X. J. Zhu, "Semi-supervised learning literature survey," 2005.

[51] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[52] G. Tur, D. Hakkani-TÃijr and R. E. Schapire, "Combining active and semi-supervised learning for spoken language understanding," *Speech Communication*, vol. 45, no. 2, pp. 171–186, 2005.

[53] X. Zhu, J. Lafferty and Z. Ghahramani, "Combining active learning and semi-supervised learning using gaussian fields and harmonic functions," in *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, vol. 3, 2003.

[54] M. F. A. Hady and F. Schwenker, "Combining committee-based semi-supervised learning and active learning," *Journal of Computer Science and Technology*, vol. 25, no. 4, pp. 681–698, 2010.

[55] Y. Leng, X. Xu and G. Qi, "Combining active learning and semi-supervised learning to construct svm classifier," *Knowledge-Based Systems*, vol. 44, pp. 121–131, 2013.

[56] M. Gao, Z. Zhang, G. Yu, S. Ã. ArÄśk, L. S. Davis and T. Pfister, "Consistency-based semi-supervised active learning: towards minimizing labeling cost," in *European Conference on Computer Vision*, Springer, 2020, pp. 510–526.

[57] *Out of Cite, Out of Mind: The Current State of Practice, Policy, and Technology for the Citation of Data*. CODATA-ICSTI Task Group on Data Citation Standards and Practices, 2013, vol. 12, pp. 1–67.

[58] FORCE-11, *Data Citation Synthesis Group: Joint Declaration of Data Citation Principles*, M. Martone, Ed. FORCE11, San Diego, CA, USA, 2014.

[59] J. Klump, R. Huber and M. Diepenbroek, "DOI for Geoscience Data – How Early Practices Shape Present Perceptions," *Earth Science Inform.*, pp. 1–14, 2015.

[60] N. Simons, "Implementing DOIs for Research Data," *D-Lib Magazine*, vol. 18, no. 5/6, 2012.

[61] J. Brase, I. Sens and M. Lautenschlager, "The Tenth Anniversary of Assigning DOI Names to Scientific Data and a Five Year History of DataCite," *D-Lib Magazine*, vol. 21, no. 1/2, 2015.

[62] "DataCite Metadata Schema Documentation for the Publication and Citation of Research Data, v4.0," DataCite Metadata Working Group, Technical Report, 2016.

[63] J. Harrow, A. Frankish, J. M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B. L. Aken, D. Barrell, A. Zadissa, S. Searle *et al.*, "Gencode: the reference human genome annotation for the encode project," *Genome research*, vol. 22, no. 9, pp. 1760–1774, 2012.

[64] D. S. Himmelstein, A. Lizee, C. Hessler, L. Brueggeman, S. L. Chen, D. Hadley, A. Green, P. Khankhanian and S. E. Baranzini, "Systematic integration of biomedical knowledge prioritizes drugs for repurposing," *Elife*, vol. 6, 2017.

[65] J. McEntyre, U. Sarkans and A. Brazma, "The BioStudies database," *Molecular systems biology*, vol. 11, no. 12, p. 847, 2015.

[66] P. Buneman, S. B. Davidson and J. Frew, "Why data citation is a computational problem," *Communications of the ACM (CACM)*, vol. 59, no. 9, pp. 50–57, 2016.

[67] S. B. Davidson, P. Buneman, D. Deutch, T. Milo and G. Silvello, "Data citation: A computational challenge," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, 2017, pp. 1–4.

[68] A. Alawini, S. B. Davidson, W. Hu and Y. Wu, "Automating data citation in citedb," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1881–1884, 2017.

[69] A. Alawini, S. Davidson, G. Silvello, V. Tannen and Y. Wu, "Data citation: a new provenance challenge," *Data Engineering*, p. 27, 2018.

[70] P. Buneman, G. Christie, J. A. Davies, R. Dimitrellou, S. D. Harding, A. J. Pawson, J. L. Sharman and Y. Wu, "Why data citation isn't working, and what to do about it," *Database*, vol. 2020, 2020.

[71] A. Y. Halevy, "Answering queries using views: a survey," *The VLDB Journal*, vol. 10, no. 4, pp. 270–294, 2001.

[72] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," in *Proceedings of the ninth annual ACM symposium on Theory of computing*, ACM, 1977, pp. 77–90.

[73] S. Chaudhuri, R. Krishnamurthy, S. Potamianos and K. Shim, "Optimizing queries with materialized views," in *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, IEEE, 1995, pp. 190–200.

[74] R. Pottinger and A. Y. Levy, "A scalable algorithm for answering queries using views," in *VLDB*, 2000, pp. 484–495.

[75] F. N. Afrati, C. Li and J. D. Ullman, "Using views to generate efficient evaluation plans for queries," *Journal of Computer and System Sciences*, vol. 73, no. 5, pp. 703–724, 2007.

[76] S. Cohen, W. Nutt and Y. Sagiv, "Deciding equivalences among conjunctive aggregate queries," *Journal of the ACM (JACM)*, vol. 54, no. 2, p. 5, 2007.

[77] S. Cohen, W. Nutt and A. Serebrenik, "Rewriting aggregate queries using views," in *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 1999, pp. 155–166.

[78] D. Srivastava, S. Dar, H. V. Jagadish and A. Y. Levy, "Answering queries with aggregation using views," in *VLDB*, vol. 96, 1996, pp. 318–329.

[79] C. Galindo-Legaria and M. Joshi, "Orthogonal optimization of subqueries and aggregation," in *ACM SIGMOD Record*, ACM, vol. 30, 2001, pp. 571–581.

[80] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh and M. Urata, "Answering complex SQL queries using automatic summary tables," in *ACM SiGMOD Record*, ACM, vol. 29, 2000, pp. 105–116.

[81] S. Cohen, W. Nutt and Y. Sagiv, "Rewriting queries with arbitrary aggregation functions using views," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 2, pp. 672–715, 2006.

[82] S. Cohen, "User-defined aggregate functions: bridging theory and practice," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ACM, 2006, pp. 49–60.

[83] C. Dwork, "Differential privacy: a survey of results," in *International conference on theory and applications of models of computation*, Springer, 2008, pp. 1–19.

[84] A. Ghorbani, A. Abid and J. Zou, "Interpretation of neural networks is fragile," *arXiv preprint arXiv:1710.10547*, 2017.

[85] M. Birattari, G. Bontempi and H. Bersini, "Lazy learning meets the recursive least squares algorithm," in *Advances in neural information processing systems*, 1999, pp. 375–381.

[86] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[87] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 463–480.

[88] S. Schelter, "Âăijamnesiaâăi–towards machine learning models that can forget user data very fast," in *1st International Workshop on Applied AI for Database Systems and Applications (AIDBâĂŹ19)*, 2019.

[89] N. A. Syed, S. Huan, L. Kah and K. Sung, "Incremental learning with support vector machines," 1999.

[90] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in neural information processing systems*, 2001, pp. 409–415.

[91] A. Deshpande and S. Madden, "Mauvedb: supporting model-based user views in database systems," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ACM, 2006, pp. 73–84.

[92] P. Gupta, N. Koudas, E. Shang, R. Johnson and C. Zuzarte, "Processing analytical workloads incrementally," *arXiv preprint arXiv:1509.05066*, 2015.

[93] A. Ginart, M. Guan, G. Valiant and J. Y. Zou, "Making ai forget you: data deletion in machine learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 3513–3526.

[94] L. Graves, V. Nagisetty and V. Ganesh, "Amnesiac machine learning," *arXiv preprint arXiv:2010.10981*, 2020.

[95] Y. Wu, E. Dobriban and S. B. Davidson, "Deltagrad: rapid retraining of machine learning models," *ICML*, 2020.

[96] J. E. Dennis Jr and J. J. MorÃľ, "Quasi-newton methods, motivation and theory," *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[97] R. Fletcher, "A new approach to variable metric algorithms," *The computer journal*, vol. 13, no. 3, pp. 317–322, 1970.

[98] H. Matthies and G. Strang, "The solution of nonlinear finite element equations," *International journal for numerical methods in engineering*, vol. 14, no. 11, pp. 1613–1626, 1979.

[99] R. H. Byrd, P. Lu, J. Nocedal and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[100] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[101] X. Wu, M. Fredrikson, S. Jha and J. F. Naughton, "A methodology for formalizing model-inversion attacks," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, IEEE, 2016, pp. 355–370.

[102] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[103] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[104] J. M. Hellerstein, C. RÃĺ, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li *et al.*, "The madlib analytics library," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, 2012.

[105] S. S. Sandha, W. Cabrera, M. Al-Kateb, S. Nair and M. Srivastava, "In-database distributed machine learning: demonstration using teradata sql engine," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, 2019.

[106] M. Schleich, D. Olteanu and R. Ciucanu, "Learning linear regression models over factorized joins," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 3–18.

[107] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu and M. Schleich, "Ac/dc: in-database learning thunderstruck," in *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, 2018, pp. 1–10.

[108] M. Schleich, D. Olteanu, M. Abo Khamis, H. Q. Ngo and X. Nguyen, "A layered aggregate engine for analytics workloads," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1642–1659.

[109] X. Zhou, C. Chai, G. Li and J. Sun, "Database meets artificial intelligence: a survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[110] S. C. Hoi, D. Sahoo, J. Lu and P. Zhao, "Online learning: a comprehensive survey," *arXiv preprint arXiv:1802.02871*, 2018.

[111] M. Dolatshah, M. Teoh, J. Wang and J. Pei, "Cleaning crowdsourced labels using oracles for statistical classification," *Proceedings of the VLDB Endowment*, vol. 12, no. 4,

[112] L. Aguilar Melgar, D. Dao, S. Gan, N. M. GÃijrel, N. Hollenstein, J. Jiang, B. KarlaÅą, T. Lemmin, T. Li, Y. Li *et al.*, "Ease. ml: a lifecycle management system for machine learning," in *11th Annual Conference on Innovative Data Systems Research (CIDR 2021)(virtual)*, CIDR, 2021.

[113] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. GÃijrel, B. Li, C. Zhang, D. Song and C. J. Spanos, "Towards efficient data valuation based on the shapley value," in *The 22nd International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 1167–1176.

[114] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang and M. Sugiyama, "Co-teaching: robust training of deep neural networks with extremely noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8527–8537.

[115] X. Zhang, X. Zhu and S. Wright, "Training set debugging using trusted items," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[116] Y. Zhang, J. Wen, X. Wang and Z. Jiang, "Semi-supervised learning combining co-training with active learning," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2372–2378, 2014.

[117] M. Nashaat, A. Ghosh, J. Miller, S. Quader, C. Marston and J.-F. Puget, "Hybridization of active learning and data programming for labeling large industrial datasets," in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 46–55.

[118] M. Nashaat, A. Ghosh, J. Miller and S. Quader, "Asterisk: generating large training datasets with automatic active supervision," *ACM Transactions on Data Science*, vol. 1, no. 2, pp. 1–25, 2020.

[119] D. S. Himmelstein and S. E. Baranzini, "Heterogeneous network edge prediction: a data integration approach to prioritize disease-associated genes," *PLoS computational biology*, vol. 11, no. 7, e1004259, 2015.

[120] Y. Amsterdamer, D. Deutch, T. Milo and V. Tannen, "On provenance minimization," *ACM Transactions on Database Systems (TODS)*, vol. 37, no. 4, p. 30, 2012.

[121] D. Dueck and B. J. Frey, "Non-metric affinity propagation for unsupervised image categorization," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, IEEE, 2007, pp. 1–8.

[122] P. Slavik, "A tight analysis of the greedy algorithm for set cover," *Journal of Algorithms*, vol. 25, no. 2, pp. 237–276, 1997.

[123] F. N. Afrati, C. Li and J. D. Ullman, "Using views to generate efficient evaluation plans for queries," *Journal of Computer and System Sciences*, vol. 73, no. 5, pp. 703–724, 2007.

[124] B. S. Arab, S. Feng, B. Glavic, S. Lee, X. Niu and Q. Zeng, "Gprom-a swiss army knife for your provenance needs," *Data Eng. Bull*, vol. 41, no. 1, pp. 51–62, 2018.

[125] A. Alawini, S. Davidson, S. Pandey, G. Silvello and Y. Wu, *"DBLP-NSF dataset SQL dump", Mendeley Data, v5.*

[126] A. Weiser and S. E. Zarantonello, "A note on piecewise linear and multilinear table interpolation in many dimensions," *Mathematics of Computation*, vol. 50, no. 181, pp. 189–196, 1988.

[127] H. Karimi, J. Nutini and M. Schmidt, "Linear convergence of gradient and proximal-gradient methods under the polyak- lojasiewicz condition," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2016, pp. 795–811.

[128] J. She and M. Schmidt, "Linear convergence and support vector identifiation of sequential minimal optimization," in *10th NIPS Workshop on Optimization for Machine Learning*, 2017, p. 5.

[129] R. Kumar and M. Schmidt, "Convergence rate of expectation-maximization," in *10th NIPS Workshop on Optimization for Machine Learning*, 2017.

[130] M. Schmidt, "Convergence rate of stochastic gradient with constant step size," 2014.

[131] L. Bottou, F. E. Curtis and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[132] H. Ning, W. Xu, Y. Chi, Y. Gong and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, vol. 43, no. 1, pp. 113–127, 2010.

[133] E. Jones, T. Oliphant, P. Peterson *et al.*, *SciPy: open source scientific tools for Python*, 2001–.

[134] M. Nikolic, M. Elseidy and C. Koch, "LINVIEW: incremental view maintenance for complex analytical queries," in *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, 2014, pp. 253–264.

[135] S. Hasani, S. Thirumuruganathan, A. Asudeh, N. Koudas and G. Das, "Efficient construction of approximate ad-hoc ml models through materialization and reuse," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1468–1481, 2018.

[136] J. Darzentas, "Problem complexity and method efficiency in optimization," *Journal of the Operational Research Society*, vol. 35, no. 5, pp. 455–455, 1984.

[137] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[138] Y. A. LeCun, L. Bottou, G. B. Orr and K.-R. MÃijller, "Efficient backprop," in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 9–48.

[139] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008, vol. 105.

[140] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[141] L. Bottou, F. E. Curtis and J. Nocedal, "Optimization methods for large-scale machine learning," *arXiv preprint arXiv:1606.04838*, 2016.

[142] J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*. Siam, 1970, vol. 30.

[143] A. R. Conn, N. I. Gould and P. L. Toint, "Convergence of quasi-newton matrices generated by the symmetric rank one update," *Mathematical programming*, vol. 50, no. 1-3, pp. 177–195, 1991.

[144] A. R. Conn, N. I. Gould and P. L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables," *Mathematics of computation*, vol. 50, no. 182, pp. 399–430, 1988.

[145] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[146] J. A. Blackard and D. J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," *Computers and electronics in agriculture*, vol. 24, no. 3, pp. 131–151, 1999.

[147] P. Baldi, P. Sadowski and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, p. 4308, 2014.

[148] D. D. Lewis, Y. Yang, T. G. Rose and F. Li, "Rcv1: a new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

[149] S. Sukhbaatar and R. Fergus, "Learning from noisy labels with deep neural networks," *arXiv preprint arXiv:1406.2080*, vol. 2, no. 3, p. 4, 2014.

[150] J. Martens, "Deep learning via hessian-free optimization.," in *ICML*, vol. 27, 2010, pp. 735–742.

[151] C. D. Meyer, *Matrix analysis and applied linear algebra*. Siam, 2000, vol. 71.

[152] G. H. Golub and H. A. Van der Vorst, "Eigenvalue computation in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1-2, pp. 35–65, 2000.

[153] A. Johnson *et al.*, "Alistair johnson, matt lungren, yifan peng, zhiyong lu, roger mark, seth berkowitz, steven horng,"

[154] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *Jama*, vol. 316, no. 22, pp. 2402–2410, 2016.

[155] A. E. Johnson, T. J. Pollard, N. R. Greenbaum, M. P. Lungren, C.-y. Deng, Y. Peng, Z. Lu, R. G. Mark, S. J. Berkowitz and S. Horng, "Mimic-cxr-jpg, a large publicly available database of labeled chest radiographs," *arXiv preprint arXiv:1901.07042*, 2019.

[156] M. Raghu, C. Zhang, J. Kleinberg and S. Bengio, "Transfusion: understanding transfer learning for medical imaging," *arXiv preprint arXiv:1902.07208*, 2019.

[157] B. Loni, L. Y. Cheung, M. Riegler, A. Bozzon, L. Gottlieb and M. Larson, "Fashion 10000: an enriched social image dataset for fashion and clothing," in *Proceedings of the 5th acm multimedia systems conference*, 2014, pp. 41–46.

[158] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "Bert: pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[159] P. Varma and C. RÃľ, "Snuba: automating weak supervision to label training data," in *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, NIH Public Access, vol. 12, 2018, p. 223.

[160] 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*, 2020, pp. 1717–1732.

[161] O. Chatterjee, G. Ramakrishnan and S. Sarawagi, "Data programming using continuous and quality-guided labeling functions," *arXiv preprint arXiv:1911.09860*, 2019.

[162] A. P. Brady, "Error and discrepancy in radiology: inevitable or avoidable?" *Insights into imaging*, vol. 8, no. 1, pp. 171–182, 2017.

[163] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[164] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard, "Handwritten digit recognition: applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 41–46, 1989.

[165] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[166] M. Belkin, D. Hsu, S. Ma and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15 849–15 854, 2019.

[167] S. S. Du, X. Zhai, B. Poczos and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *International Conference on Learning Representations*, 2018.

[168] Z Allen-Zhu, Y Li and Y Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," *Advances in neural information processing systems*, 2019.

[169] S. Du, J. Lee, H. Li, L. Wang and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 1675–1685.

[170] S. Arora, S. Du, W. Hu, Z. Li and R. Wang, "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 322–332.

[171] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar and N. Srebro, "The implicit bias of gradient descent on separable data," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2822–2878, 2018.

[172] L. Chizat and F. Bach, "Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss," in *Conference on Learning Theory*, PMLR, 2020, pp. 1305–1338.

[173] S. Gunasekar, J. D. Lee, N. Srebro and D. Soudry, "Implicit bias of gradient descent on linear convolutional networks," *Advances in Neural Information Processing Systems*, vol. 2018, pp. 9461–9471, 2018.

[174] M. Kubo, R. Banno, H. Manabe and M. Minoji, "Implicit regularization in over-parameterized neural networks," *arXiv preprint arXiv:1903.01997*, 2019.

[175] L. Zhu and S. Han, "Deep leakage from gradients," in *Federated Learning*, Springer, 2020, pp. 17–31.

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

[177] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[178] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2016.

[179] A. Bietti and J. Mairal, "On the inductive bias of neural tangent kernels," in *NeurIPS 2019-Thirty-third Conference on Neural Information Processing Systems*, vol. 32, 2019, pp. 12 873–12 884.

[180] L. Rupprecht, J. C. Davis, C. Arnold, Y. Gur and D. Bhagwat, "Improving reproducibility of data science pipelines through transparent provenance capture," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3354–3368, 2020.

[181] J. Hu, J. Joung, M. Jacobs, K. Z. Gajos and M. I. Seltzer, "Improving data scientist efficiency with provenance," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, IEEE, 2020, pp. 1086–1097.

[182] J. F. N. Pimentel, V. Braganholo, L. Murta and J. Freire, "Collecting and analyzing provenance on interactive notebooks: when ipython meets noworkflow," in *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[183] T. Kraska, A. Beutel, E. H. Chi, J. Dean and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 489–504.

[184] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," *arXiv preprint arXiv:1808.03196*, 2018.

[185] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan and I. Stoica, "Deep unsupervised cardinality estimation," *Proceedings of the VLDB Endowment*, vol. 13, no. 3, pp. 279–292, 2019.

[186] V. Trkulja and P. Hrabač, "Confidence intervals: what are they to us, medical doctors?" *Croatian medical journal*, vol. 60, no. 4, p. 375, 2019.

[187] G. A. Kaissis, M. R. Makowski, D. RÃijckert and R. F. Braren, "Secure, privacy-preserving and federated machine learning in medical imaging," *Nature Machine Intelligence*, vol. 2, no. 6, pp. 305–311, 2020.

[188] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: a natural synergy," in *Proceedings of the 2018 international conference on management of data*, 2018, pp. 1645–1650.

[189] H. Mousavi, S. Gao and C. Zaniolo, "Ibminer: a text mining tool for constructing and populating infobox databases and knowledge bases," *Proceedings of the VLDB Endowment*, vol. 6, no. 12, pp. 1330–1333, 2013.

[190] Y. Li, B. Rubinstein and T. Cohn, "Exploiting worker correlation for label aggregation in crowdsourcing," in *International Conference on Machine Learning*, PMLR, 2019, pp. 3886–3895.

[191] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger and S. Friedler, "Problems with shapley-value-based explanations as feature importance measures," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5491–5500.

[192] "Explainable artificial intelligence: a survey," in *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, IEEE, 2018, pp. 0210–0215.

[193] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester and L. De Raedt, "Deepproblog: neural probabilistic logic programming," *Advances in Neural Information Processing Systems*, vol. 31, pp. 3753–3760, 2018.

[194] A. d. Garcez, T. R. Besold, L. De Raedt, P. FÃűldiak, P. Hitzler, T. Icard, K.-U. KÃijhnberger, L. C. Lamb, R. Miikkulainen and D. L. Silver, "Neural-symbolic learning and reasoning: contributions and challenges," in *2015 AAAI Spring Symposium Series*, 2015.