Bachelor's Programme in Science and Technology

Data Lineage and Observability for Ensemble Machine Learning Serving in the Edge

Dung Nguyen Anh

Aalto University

BACHELOR'S THESIS

Aalto University
BACHELOR'S THESIS 2023

Data Lineage and Observability for Ensemble Machine Learning Serving in the Edge

Dung Nguyen Anh

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Technology.
Otaniemi, 28 Apr 2023

Supervisor: professor Maarit Korpi-Lagg
Advisor: Minh-Tri Nguyen

Aalto University
School of Science
Bachelor's Programme in Science and Technology

Author
Dung Nguyen Anh

Title
Data Lineage and Observability for Ensemble Machine Learning Serving in the Edge

School School of Science

Degree programme Bachelor's Programme in Science and Technology

Major Data Science **Code** SCI3095

Supervisor professor Maarit Korpi-Lagg

Advisor Minh-Tri Nguyen

Level Bachelor's thesis **Date** 28 Apr 2023 **Pages** 22+3 **Language** English

Abstract

With the development of machine learning as a service (MLaaS), the demand for more explainable services is increasing. To match this requirement, ML providers utilize monitoring systems that capture multiple ML-specific attributes to verify the alignment with the service level agreement (SLA). However, because the current state of monitoring tools can only capture the performance of individual models, their relationship with the ensemble model remains obscure to both ML provider and their customers. Moreover, dynamic inference systems employing autoscaling or auto ensemble techniques have introduced uncertainty into the inference process, decreasing the observability of the ML system. This paper proposed incorporating data lineage in the monitoring system to increase model observability and capture essential data for inference explanation. To achieve that, we discuss the applicability of PROV-DM as the standard for representing data lineage in the ML domain, which is employed as the foundation of the proposed library and database schema. The suggested solution can automatically capture the inference process with minor code modifications and runtime overhead. To illustrate the application of the library, we conduct an experiment that analyzes the runtime overhead of the library and two scenarios where data lineage can help improve ML service.

Keywords keyword

urn https://aaltodoc.aalto.fi

Contents

1. Introduction

In recent years, Machine Learning (ML) has been applied extensively to many problems, including financial forecasting [1], image recognition [2], and natural language processing [3]. With their proven performance and the increasing number of ML applications, several organizations seek third-party ML service providers to improve their operations, thus leading to the emergence of a new business model — ML as a Service (MLaaS). Like other services, MLaaS requires a contract called a service level agreement (SLA) that sets the expectations and describes the delivered service to preserve the benefits of both stakeholders [4]. However, the current SLA for MLaaS has yet to fulfill its preservation job because it is challenging to measure various ML-specific attributes, such as data quality, inference accuracy, and explainability, while comprehensive end-to-end observability of those attributes is essential in monitoring contract [5]. For instance, in Internet of Things (IoT) applications, the quality of data causes many issues with the performance of ML solutions as the data in IoT can be impacted by many factors such as erroneous measurement, environmental noise, and discrete observations [6]. Then, as the accuracy of the ML model depends on data quality, they can produce unreliable predictions when data quality problems exist, causing ML providers to be penalized based on the SLA. Another issue is the transparency of the inference systems because, in MLaaS, the customers only submit the data and do not cooperate with the ML provider in the prediction task. Although a level of abstraction is beneficial as the customer does not require a complete understanding of the system, a complete black box model where the inference process is obscure is not desirable, especially for critical applications like e-health or autonomous vehicles.

Increasing model explainability by interpreting the inference result to the customer is one approach to mitigate the challenges caused by the black box characteristics and the relationship between data quality and ML service performance [7]. However, as most of the recent work addresses explainability aspects in the

training phase [6], it is unclear about the appropriate utilization and implementation of such attributes in the serving phase and ML service contract. To support ML-specific SLA, Linh et al. [5] proposed the QoA4ML framework which outlines the essential components of such contracts, the definition of ML attributes and constraints, and the guidance on the process to monitor and assess these elements. Although QoA4ML has created a foundation for robust ML monitoring, the current implementation only focuses on monitoring metrics of each base model. Thus, as it is challenging to monitor and explain the connections between base models in an ensemble model and between the ensemble models in the whole systems with the current state of QoA4ML, they remain a black box to both stakeholders. Such a level of explainability is insufficient for some sophisticated applications, such as digital assistants, which employ a complex chain of models. Additionally, the increasing research in autoscaler for the deep learning inference serving system [8] and automated ensemble [9] demands a new solution to capture inference graphs to ensure dynamic and robust inference capabilities of ML solutions.

This thesis explores a novel approach that helps increase the observability of ML systems by capturing data lineage. Firstly, two data models for representing data lineage are discussed to prove PROV-DM as a more appropriate selection for capturing data lineage in ML applications. Then, with PROV-DM as the foundation, a prototype library along with a database schema for data lineage is implemented, which can automatically record the inference process of ensemble models while preserving the overall performance of the system. Lastly, the proposed solution is utilized in three experiments to demonstrate how it can help improve the current state of MLaaS.

The rest of this paper is organized as follows. Section 2 discusses the background of the research, and Section 3 explains the reason for choosing PROV-DM and its specific implementation to capture data lineage in ensemble ML serving. Section 4 describes the experiment and its result, while Section 5 concludes the research.

2. Background

In this section, the first part presents the definition of ensemble ML, its popular applications in different fields, and its explainability challenge. Then, Section 2.2 examines the description of data lineage and the way it has been utilized to increase the explainability of many systems. Lastly, Section 2.3 discusses one standard for the data model of provenance used in Section 3, while Section 2.4 considers the related works.

2.1 Ensemble ML and its Explainability Challenges

Ensembling is a conventional technique that aggregates multiple models, which can reduce the prediction variance, to improve the accuracy and robustness of predictions [10]. This advantage is achieved by utilizing multiple base models with different algorithms, hyperparameters, or subsets of features on the same input dataset and then combining their results by applying various aggregation methods such as averaging, voting, or stacking to produce the final decision. Such a technique is highly effective in many applications, including financial forecasting [1], image recognition [2], and natural language processing [3].

However, dataflow graphs are becoming more complex with the growing demand for more accurate and advanced predictions. These graphs are typically represented as a directed acyclic graph (DAG) and consist of multiple base models that are ensembled, interconnected, and orchestrated to, sequentially or in parallel, handle the input data[8]. For each prediction request, the whole DAG or only a subset of it can be involved, requiring ML providers to record the data flow of the inference task for further analysis and explanation. Besides that, monitoring the data pipeline is crucial because although the ensemble approach enhances the robustness of the final prediction, it still depends heavily on its other models. Thus, when the performance of some, or even one, models declines because of data quality problems, the quality of ensembled prediction will also decrease,

potentially resulting in the violation of the SLA.

In addition, other challenges have appeared from new techniques in recent years. For instance, in resource-constrained environments like the edge, Bai et al., [9] propose an ensemble selection algorithm that only employs models with the best inference performance. Although this approach can help resolve the challenges of high computation complexity, high resource occupation, and the average inference time of ensemble ML, it introduces uncertainty to the inference flow, making the data pipeline less transparent and explainable. Another technique proposed by Razavi et al. [8] can efficiently automatically scale deep learning inference serving systems to align with strict SLA on the end-to-end latency. Those two automated techniques are crucial components of a dynamic inference system that is challenging for the current interpreting techniques to explain without capturing the data lineage of the ML model.

2.2 Supporting Explainability with Data Lineage

The terms lineage, provenance, and traceability can be used interchangeably to refer to the process of constructing a final product, whether a digital file or a physical item [11]. Regarding lineage for the digital file, Wang et al. [12] was the first paper to discuss those issues, which helped formally define provenance as “data source tagging” and “intermediate source tagging” problems [13, p. 5]. From its original interpretation, provenance has been applied in many domains such as scientific databases [14] [15] [16], data warehouses [17] [18], and recently big data platforms [19] [20], and IoT [21]. In different applications, the types of lineages, their generating techniques, and the main issue they can answer are diverse [11]. For instance, there are four main provenance classes, i.e., data provenance, information systems provenance, provenance meta-data, and workflow provenance.

With the development of many AI applications, researchers have tried to incorporate provenance into the AI systems [22] by linking the input and output of the model, which can be a valuable source in interpreting the inference result. However, in this domain, provenance is commonly utilized in a rather general way, where entire algorithms or data transformations are represented by semantic relationships [23]. Consequently, while entire pipelines can be documented with provenance, the specific inner workings of individual models remain opaque, which is insufficient to explain the dynamicity of the serving system.

Additionally, provenance is often considered when assessing data quality tasks, e.g., evaluating the integrity, trust, and accuracy of the data. By analyzing

PROV Concepts	Classes	Name
Entity	PROV-DM types	Entity
Activity		Activity
Agent		Agent
Generation	PROV-DM relations	WasGeneratedBy
Usage		Used
Communication		WasInformedBy
Derivation		WasDerivedFrom
Attribution		WasAttributedTo
Association		WasAssociatedWith
Delegation		ActedOnBehalfOf

Table 2.1. PROV-DM core concepts and types [26]

provenance, the AI systems can detect errors in data generation and processing, which is valuable for IoT applications where data is uncertain, erroneous, and noisy [24]. Then, when combined with another monitoring service like Prometheus [25], ML providers can moderately explain the relationship between the quality of data and quality of inference, helping them to establish appropriate contracts that define the tolerable level of data quality for the ML service to function decently [5].

2.3 Standard for Representing Data Lineage

Researchers have proposed various models, languages, and tools to facilitate the documentation of provenance, including those tailored for AI/ML models [27]. However, it is still crucial to have a standard reusable ontology to capture provenance in heterogeneous environments, such as the web, to develop provenance documentation further. As an approach to resolve this problem, the PROV family of documents (PROV-OVERALL) was introduced by The World Wide Web Consortium (W3C) in 2013 [26]. In those documents, the most significant is the PROV data model (PROV-DM), which provides a universal data model for provenance that can be applied to translate domain-specific and application-specific representations of provenance into a standardized format that can be shared between different systems. PROV-DM has ten core concepts which are divided into types and relations like in Table 2.1. As illustrated in Figure 2.1, PROV-DM outlines the employment and production of *entities* by *activities*, which may be influenced in several manners by *agent*. The following paragraphs will discuss the proper definition and application of those data models.

Firstly, entities are the fundamental building blocks of the PROV-DM data model. They represent physical or digital objects involved in a process, such as a document, dataset, or prediction result. Entities can have fixed attributes describing their characteristics, such as size, quality, or creation time. Then, when the value of an

attribute is adjusted, a new entity is produced and related to the previous version ("wasDerivedFrom").

Secondly, activities are actions or processes that transform or manipulate entities. For instance, activities include data processing, predicting, and request serving. Activities can have attributes describing their properties, such as the start and end times of the task or the ML models employed to generate the prediction. Then, this data model can be "used" by entities as their inputs, and it can output a new entity which "wasGeneratedBy" the activities. Moreover, an activity can be informed by other activities with "wasInformedBy" relations.

Lastly, agents are things that are responsible for executing activities ("wasAssociatedWith"). When one activity is finished, an entity is produced, which is attributed back to the agent ("wasAttributedTo"). Agents can be people, software tools, ML models, or other entities that can initiate or control activities. Additionally, an agent can "actedOnBehalfOf" other agents, describing their delegation relationship.

With the above definition of PROV-DM, Section 3 compares it with another provenance data model to discuss the reason for choosing it as the primary data model to capture the data lineage of ML applications.

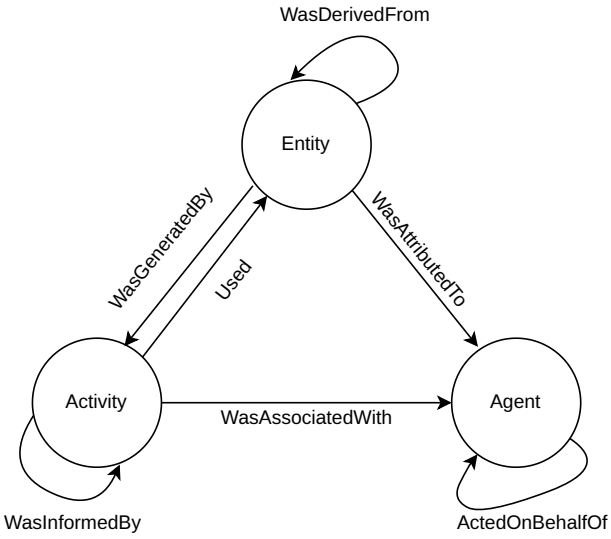

Figure 2.1. W3C PROV-DM structure [26]

2.4 Related Work

Data lineage/provenance for data science: In the data science world, the need for capturing data provenance is increasing as it enables trust and provides users with tools that allow them to access, record, and further investigate resources and steps in a workflow [27]. Then, to represent data provenance in different

applications, PROV-DM is a popular data model that many researchers utilize. For instance, [22] is a system that adopts design principles to provide efficient distributed data capture from the workflow, while [28] is a provenance service designed to support online hyperparameters analysis in Deep Neural Networks. However, those proposed systems are not applicable to ML serving as essential qualities for service explainability, such as data quality and prediction quality, are not automatically captured. Our work introduces a library that can capture the data lineage in ML serving for observability and essential attributes for service explanation.

3. Data Lineage for Ensemble ML Serving

3.1 PROV-DM Comparison With Another Approach

One alternative to PROV-DM is the open provenance model (OPM) [29], which was first released in 2007 as a result of the Provenance Challenges series in 2006. This model represents the provenance graph with a DAG which contains three types of vertices: artifact, process, and agent. While the *artifact* is the 'immutable piece of state' that portrays the focused entities of provenance, the *process* illustrates the course of actions that are enabled, facilitated, controlled, and affected by the *agent* [29, p. 3]. In the provenance graph of the OPM, the edges describe the dependencies and causal relationship between the entities, such as the generated by, triggered by, or used relation. From these descriptions, we can recognize that the OPM created a foundation for the later development of PROV-DM as most of their entities and relationship can be directly mapped to the other according to Table 3.1. Then, with less relationship, the OPM is a more lightweight model that can be employed to present provenance information of ML applications. However, some aspects of PROV-DM are more beneficial than the OPM, and the following paragraphs discuss these advantages.

Firstly, while OPM was initially developed for scientific workflows, PROV-DM is a domain-agnostic data model which can be applied and extended to describe provenance data in various fields. For its extensibility, PROV-DM is more suitable to capture the data lineage of ML as there are numerous applications of ML. For example, Souza et al. [22] included new *referred* and *hadStore* relationships to represent data references in a heterogeneous database, while Pina et al. [28] proposed a new domain-specific data model based on PROV-DM to represent training-specific data from deep learning experiments called DNNProv-Df.

Secondly, as OPM is a generic provenance model, its granularity level is lower than PROV-DM. Specifically, while OPM can only describe causal relationships,

OPM	PROV
Artifact	Entity
Process	Activity
Agent	Agent
Used	Used
WasGeneratedBy	WasGeneratedBy
WasTriggeredBy	WasInformedBy
WasDerivedFrom	WasDerivedFrom
WasControlledBy	WasAssociatedWith
	ActedOnBehalfOf
	WasAttributedTo

Table 3.1. Comparison between OPM and PROV-DM

PROV-DM can represent attribution and delegation in its core and many others in its extended model. In the context of XAI, such a low level of granularity is insufficient for explaining ensemble ML which requires a decent amount of data about their base models.

Lastly, the higher level of adoption of PROV-DM compared to OPM is beneficial for the ML provider. As all stages of the ML life cycle require provenance for explainability, traceability, and replicability, a holistic data model is advantageous for provenance data analysis throughout the life cycle. For instance, an ML provider can use Keras-Prov [28] to fine-tune the hyperparameter of the model in the training phase and then employ our proposed library in the production phase. As both libraries share the same provenance data model, more data analysis can be supported and is easier to implement.

Thus, because of those three key advantages, PROV-DM is our selected data model to represent the provenance data of ML applications. Moreover, by applying it, our solution can automatically capture provenance data at run time and save it to the database for further data analysis and inference explanation. However, this research only applies parts of PROV-DM without any extension as they are sufficient to represent the most critical and popular data provenance in ML systems. The extension for specific applications can be developed and quickly integrated through the architecture in the following section.

3.2 Architecture

3.2.1 Overview and Design Principles

As the research target is data lineage in the production stage, some requirements must be incorporated into the design principle of the library. Firstly, as the ML system operates under the strict SLA regarding response time, a high runtime

overhead can cause a violation, which is intolerable for ML providers. Thus, the proposed library focuses on diminishing runtime provenance capture, which does not affect the overall performance of the applications. Secondly, given the multitude of ML applications, the library should be easily integrated into the existing solutions without too much implementation cost. Finally, the choice of the database for storing the data lineage needs to be scalable to support a large number of writing requests while fostering various kinds of data analysis. With those mentioned requirements, we implement the library following these design principles:

- Lightweight: The proposed library is compact and minimal, and it can be imported and integrated into the serving system to enable instrumentation with little code modification. Thus, this principle prevents unnecessary pollution and complexity in the original codebase. The library only contains simple methods to capture crucial activities, such as predicting, preprocessing, and ensembling, in the specific order executed in the systems. For example, the below code is an example for capturing preprocessing activity, which only requires a simple decorator above the main function. Moreover, this solution handles the data lineage insertion and related relationships into the database in the background, requiring no attention from the user.

```
@capture(activityType='preprocessing')
def enhance_image(data):
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    enhanced_im = cv2.filter2D(src=data.data, ddepth=-1, kernel=kernel)
return enhanced_im
```

- Asynchronicity: As an ML serving system processes a massive number of requests concurrently in a strict SLA, a practical solution manages the required time-consuming tasks to run asynchronously and not block the code execution. Then, the library utilizes the Python Threading module to run most of the tasks asynchronously. Although synchronous functions exist in the library, the overall performance with data lineage capturing is still guaranteed, and it is analyzed in Section 4.

- Flexibility: Data lineage can be employed in various tasks, from system monitoring to data analysis, so a database that can foster different types of analysis is preferred. Moreover, the enormous number of requests demands a scalable database that can handle the high write rate of capturing data lineage tasks. Thus, the choice of the database is the most challenging part of the research

because it requires a database that is not only flexible but also scalable.

With those principles in mind, the following sections discuss the detailed implementation of our solution.

3.2.2 PROV-DM Abstraction

With PROV-DM as the foundation, we can transform this conceptual data model into specific terms in the object-oriented programming (OOP) paradigm for efficient implementation: agent into class, activity into methods or functions, and entity as the input and output of the methods or functions. The specific implementation is discussed in the following paragraphs.

Firstly, to capture the agent, the library has a decorator `captureModel` that can wrap the `init` function of the class to automatically capture the name and parameters of the model. Then, those data are inserted into the database to identify individual models. If there are multiple replicas of that model, all of them will be recorded in the database with a unique id. This automatic model capturing can help investigate the relationship between model parameters and the predicting performance.

Secondly, the methods in different classes or individual functions are captured as an activity by decorator `capture` with `activityType` as the argument. Currently, the library focuses on four crucial tasks in a standard ML system, i.e., assessing data quality, preprocessing, predicting, and ensembling. Then, by passing the activity type to the decorator, it wraps the functions so that the necessary input and output of the task are automatically captured according to pseudo-Algorithm 1. Then, the results of the wrapped activity are transformed into different entity classes, which are asynchronously inserted into the database accordingly in a different Thread. Additionally, the decorator can calculate some standard metrics in a prediction task such as CPU or memory usage, start and end timestamp, and even the quality of analysis if the user specifies it in the result of the activity.

Lastly, an entity, the targeted product of provenance capturing, is employed to abstract the input and output of activity executed in the system. Corresponding with the four activity types above, there are four entity classes: `Data`, `DataQualityReport`, `Prediction`, and `PredictionQuality`. When the client submits their data to ML providers, it is first captured with the function `captureInputData` along with the unique `requestId` to identify the individual request in the database. Then, the output of the functions that are decorated with `capture` is transformed into the equivalent entity classes (line 4, 7, 10, 13 of pseudo-Algorithm 1), inserted asynchronously into the database, and returned to be utilized in later functions. The

decorator handles this process of transforming the output into an equivalent entity class, reducing the need for code modifications. However, because the proposed entity classes are an abstraction for the input and output of targeted activity, the variable must be taken outside the class to be employed by other functions.

Algorithm 1 Capturing Provenance Wrap Function

```
1: function CAPTURE(activityType)
2:     function WRAPPER(func)
3:         if activityType = predict then
4:             returnVal ← Prediction(func())
5:             capturePredictActivity (returnVal)
6:         else if activityType = assessDataQuality then
7:             returnVal ← DataQualityReport(func())
8:             captureAssessDataQualityActivity (returnVal)
9:         else if activityType = preprocessing then
10:            returnVal ← Data(func())
11:            capturePreprocessingActivity (returnVal)
12:        else if activityType = ensemble then
13:            returnVal ← Prediction(func())
14:            captureEnsembleActivity (returnVal)
15:        end if
16:        return returnVal
17:    end function
18:    return wrapper
19: end function
```

3.2.3 Database Choice

As the required data lineage has been captured with the provided decorator, function, and entity classes, it requires a corresponding database to store those data, which must be aligned with our design principle listed before. Thus, the relational database, specifically PostgreSQL, is selected for two reasons.

Firstly, relational databases offer high flexibility in managing and organizing data. As the data is efficiently stored in a structured manner with an explicit relationship between tables, it enables uncomplicated modifications and updates to the database schema without disrupting the existing data. This characteristic is crucial for capturing the data lineage of an ML system as there are various applications, and the proposed schema is only sufficient in specific scenarios. In other cases, it may require the modification of the database schema to capture the whole data lineage. Additionally, relational databases foster complex queries that provide data manipulation and analysis flexibility. This characteristic makes it simpler to adapt the database to changing business requirements and the dynamicity of the advanced ML system. Compared to the NoSQL database that requires extract, transform, load (ETL) process to obtain the required resilience, a

relational database is preferred.

Secondly, relational databases are scalable, making them suitable for capturing a massive amount of data lineage. Relational databases can scale vertically, which requires adding more CPU, memory, or storage, or horizontally, which utilizes sharding or partitioning to distribute the data across multiple nodes, to handle increased data loads and concurrent connection. Besides the general scalability characteristics of relational databases, the choice of PostgreSQL, an open-source database, supports a high level of performance customizability from the large ecosystem of the PostgreSQL extension. For example, in the data lineage case, TimescaleDB can be utilized to match the number of inserting queries. Although NoSQL can be scaled with much less effort than a relational database, the more flexibility after the data has been stored, and the sufficient scalability makes it a more reasonable choice.

3.2.4 Relational Database Schema

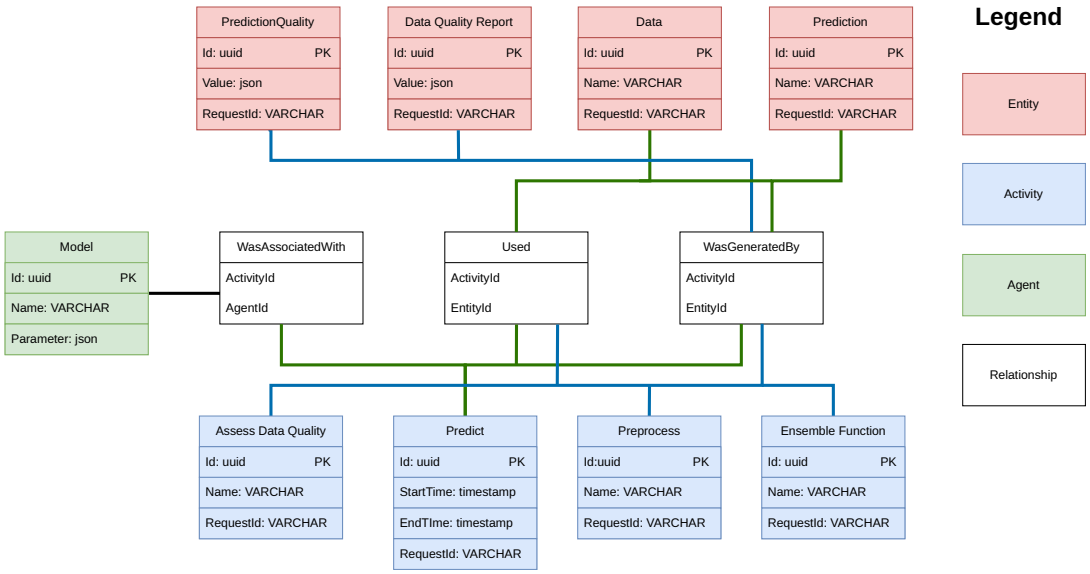

Figure 3.1. Database schema

Relational databases require a database schema that defines the organization and arrangement of data. Thus, as the solution employs PROV-DM data models, the tables in the suggested schema, illustrated in Figure 3.1, belong to one of four groups: entity, agent, activity, and relations. All the tables in entity, agent, and activity groups are directly linked with the functions and classes in the library, while the relation tables are kept intact with the PROV-DM model. However, only the three most important relationships, i.e., was associated with, used, and was generated by, are implemented as this research focuses on capturing the data

lineage for ensemble ML. Those relationship tables do not have a foreign key as different entities and activity tables have the same relationship.

Using the proposed schema, it is possible to (i) track the model parameters with its performance and resource usage, (ii) automatically trace the root cause of each false prediction, (iii) efficiently understand the impact of each base model on the ensemble function, (iv) interpret the results generated at sufficient granularity from end to end, and (v) capture the relationship between data quality and prediction quality to preserve the benefits of both stakeholders. Two of those applications will be experimented in Section 4.

4. Experiment and Results

In this section, an experiment is conducted to demonstrate the low computational overhead when integrating the library into an example ML serving system. Then, from the data lineage generated in the first experiment, the relationship between the ensemble function and its base models is discussed. Lastly, the proposed solution is further experimented with a scenario that simulates a real-world situation where there is a data quality problem. All experiments are performed utilizing the following settings.

4.1 Experiment Settings

In the following paragraphs, the experiment setup is described, including the hardware, an example serving system framework, an ensemble ML model with one sample dataset, and the PostgreSQL database with its Python adapter.

Hardware: The experiment was conducted on a local machine with the following specifications: a Ryzen 5 5600X processor, an NVIDIA GeForce RTX 3070 graphics card, 16GB DDR4 RAM, and running Pop!_OS 22.04 as the operating system. The machine was configured to meet the requirements of the experimental setup, including software installation, example dataset, and data analysis Jupyter notebook. While running the experiment, the machine is controlled with minimal background tasks, a stable power supply, and appropriate delay between runs to ensure reliability during the experiment.

Serving system framework: Ray [30] is an open-source distributed computing framework for scalable and efficient data-intensive and ML workloads. Ray provides simple programming abstractions such as actors and tasks for building complex distributed applications, making it quick to implement decent ML serving systems. Moreover, it supports concurrent and stateful programming with actors and allows for efficient parallel execution of tasks. The version of Ray that is

utilized in this experiment is 2.3.0.

Ensemble ML model: Our choice for an example ML application is object detection as it is extensively utilized with various pre-trained models. Then, Yolov5 [31] and Yolov8 [32] are selected as they are both open-source projects developed by Ultralytics, which are known for their simplicity in terms of implementation and setup. Because both models have pre-trained versions, so, for low computing power, Yolov5s and Yolov8n versions are selected. After both models have generated their prediction, they will be ensembled with a simple max aggregation function.

Sample dataset: For object detection development and evaluation, there are multiple open datasets, such as Microsoft Common Objects in Context (COCO), ImageNet, or Dataset for Object deTection in Aerial Images (DOTA). In these experiments, Google Open Image Dataset V1 is selected. Then, from 600 classes in the dataset, only five of them are chosen, and each class has 100 images that are put into different folders: cat, dog, person, car, and bicycle.

PostgreSQL and Python adapter: PostgreSQL version 15.2 with default setting is locally hosted using the Docker container. Then, Psycopg3 version 3.1 is utilized as a Python adapter because the library provides a simple and efficient way to interact with the database, both synchronously and asynchronously.

4.2 Experiments

4.2.1 Effect of Capturing Lineage on Serving Performance

In the first experiment, the deployment of the example ensemble model is operated *with* and *without* data lineage captured for handling requests from one client. A random class folder will be selected, and then a random image within that folder will be sent to the Ray cluster through an HTTP request. The response time for each request, calculated by the time the client makes the request until receiving the response, is recorded. In total, 1000 requests are sent consecutively.

The first 300 requests are ignored to mitigate potential warm-up effects or initial resource consumption variances and guarantee reliable results. With the remaining 700 requests, the mean and 99th percentile (p99) response time will be calculated to analyze the effect of capturing data lineage on the entire system. While the p99 latency will record any observable changes in the tail-end performance, the mean response time will give an overall measure of performance.

As we can see from Table 4.1, the difference in the mean response time is negligible, but there is a noticeable in the p99. The main reason is that resource

Response Time (s)	Without Lineage	With Lineage
Mean	0.3943	0.4031
Min	0.2139	0.2258
Max	0.5673	0.5802
P99	0.5537	0.5660

Table 4.1. Summary of response time with and without lineage

usage is calculated during the data lineage capturing, which cannot be executed asynchronously. Such an increase in the p99 latency is tolerable and will be much lower when applied to more powerful machines. On the other hand, the Figure 4.1, which shows the fluctuation in the rolling average of the response time, suggests that the overall performance of deployed ML systems with data lineage capturing remains almost similar to the original one. Then, although the integration of data lineage still creates some computation overhead, the library has followed the lightweight design principle and will not affect the violation rate of SLA.

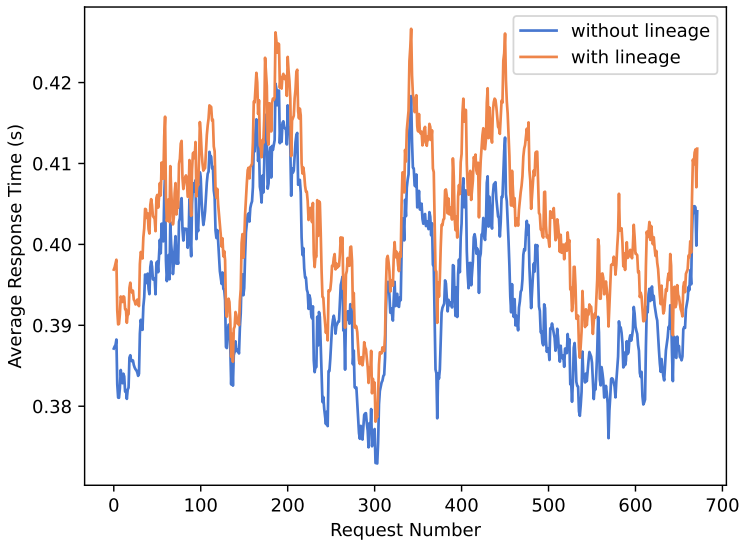

Figure 4.1. 30 request rolling average of response time

4.2.2 Data Lineage for Ensemble Model Analysis

The data lineage generated in the previous experiment can be utilized to analyze the prediction performance of the ensemble model. Firstly, a SQL script is created to query the database for all related data of the max aggregation function, which can be found in Appendix A. Then, the prediction from both base models and their aggregated ones is verified with the ground truth to determine if the prediction is correct, which is used to calculate the accuracy for different classes.

The data in Table 4.2 shows that the aggregated prediction generally outperforms separated prediction. It only accomplishes similar accuracy with its base models

Model \ Accuracy	Bicycle	Car	Cat	Dog	Person
Yolov5n	0.36	0.4	0.14	0.28	0.89
Yolov8n	0.66	0.69	0.88	0.74	0.98
Max Aggregated	0.71	0.73	0.88	0.78	0.98

Table 4.2. Accuracy of three models over five classes

when there is an enormous (0.88 vs. 0.14) or insignificant (0.98 vs. 0.89) difference in the accuracy of Yolov8n and Yolov5n. By analyzing this relationship, ML providers can make rational decisions on keeping or eliminating specific models, leading to more efficient and effective ML systems. Data lineage is crucial to facilitate this decision-making process as it provides the necessary data about the relationship between the ensemble function and its base models. Overall, using data lineage in analyzing prediction performance is elemental for improving the quality and reliability of ML services.

4.2.3 The Root Cause of Performance Decrease

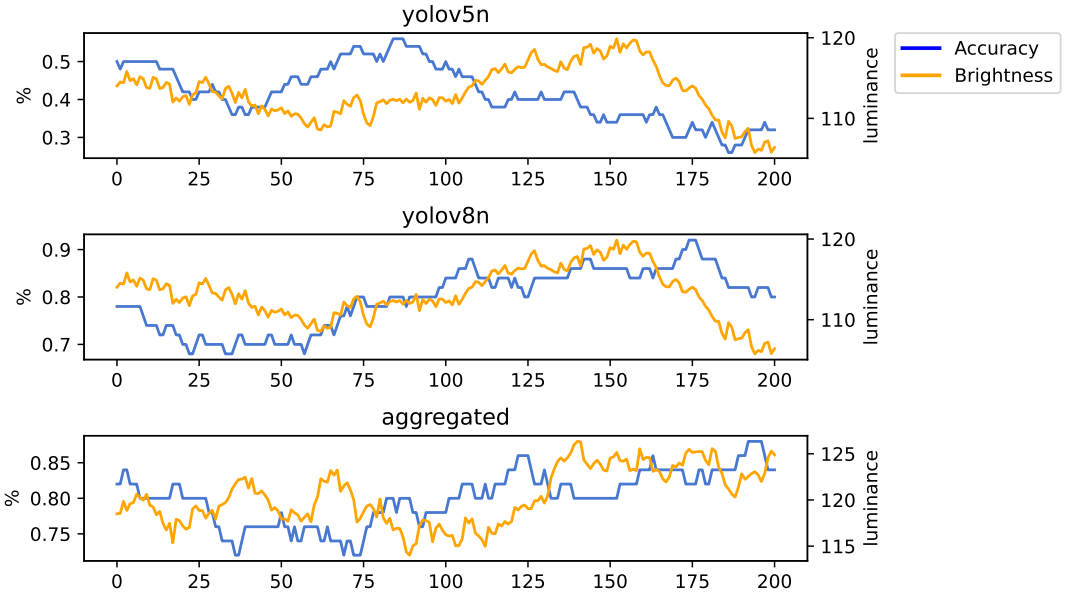

Figure 4.2. Original Accuracy

This final experiment aims to show how our solution detects the relationship between the ensemble model and their base model when data quality problems exist. To achieve this goal, we analyze the data generated in the first experiment and combine it with new data obtained by sending another 300 requests to the server with a degraded version of the example dataset, which is produced by increasing the brightness of the image with `convertScaleAbs` function in OpenCV [33] and the alpha factor of 2. The degraded dataset simulated a scenario where

the environment is sunny, which is a common situation that could affect the input image quality.

In Figure 4.2, we can see that the accuracy of all models increased as the brightness of the images decreased. However, the more significant gap between the brightness and accuracy line shows that the effect was different for all models. Moreover, when the brightness is manually adjusted to a higher level in Figure 4.3, the performance of Yolov8n decreases more than Yolov5n. This decrease resulted in a significant decrease in the performance of the aggregated function, potentially violating the SLA. To deal with this scenario, ML providers must understand the effect of different data quality on their ensemble model to set up a suitable contract that dynamically adapts to the fluctuation of data quality. The proposed solution can provide the necessary data and analysis flexibility for that task.

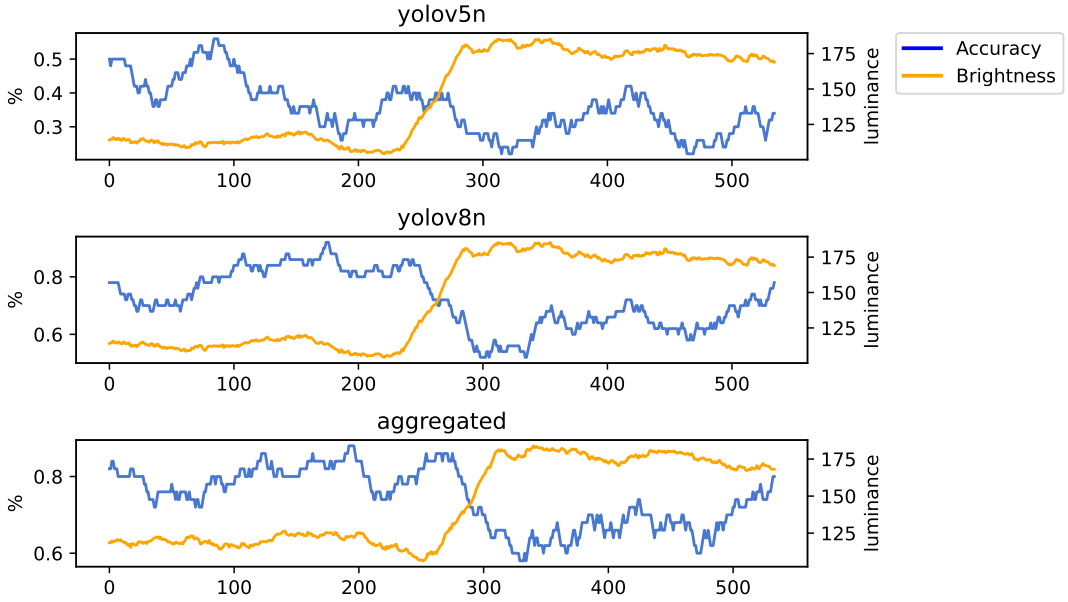

Figure 4.3. Accuracy after brightness increase

5. Conclusion

As the number of ML applications along with MLaaS continues to grow, it requires the higher observability of the whole ML system to support service explainability, which helps preserve the benefits of both stakeholders while complying with the incoming regulations. In this research, the observability challenges of ensemble ML are mitigated by applying the PROV-DM data model to create a lightweight library that can automatically capture the data lineage of the ML system without too many code modifications. Moreover, the choice of the database with the corresponding schema is proposed, supporting flexibility and moderate scalability. Then, the experiment is performed to show the effect of capturing data lineage on the performance of an example ML system and the way data lineage can be utilized to improve ML service in specific scenarios.

However, this research has not fully addressed the scalability problem of capturing data lineage as PostgreSQL can only scale moderately at its original state. Moreover, the proposed library cannot be quickly integrated with other monitoring tools, such as Prometheus or QoA4ML. In the future, a customization or extension to PostgreSQL that can handle higher write rates could be researched to handle real-world workloads. Thus, this paper has discussed that capturing the data lineage of the whole ML system is an appropriate solution to improve MLaaS, which should be researched more to be incorporated into existing monitoring tools to create a holistic observability solution for dynamic serving systems.

Bibliography

[1] S. Sun, S. Wang, and Y. Wei, "A new ensemble deep learning approach for exchange rates forecasting and trading," *Advanced Engineering Informatics*, vol. 46, p. 101160, 2020.

[2] Y. Chen, Y. Wang, Y. Gu, X. He, P. Ghamisi, and X. Jia, "Deep learning ensemble for hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 6, pp. 1882–1897, 2019.

[3] S. Das, M. Ashrafuzzaman, F. T. Sheldon, and S. Shiva, "Network intrusion detection using natural language processing and ensemble machine learning," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 829–835, 2020.

[4] D. Serrano, S. Bouchenak, Y. Kouki, F. A. d. O. Jr, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "SLA guarantees for cloud services," *Future Generation Computer Systems*, vol. 54, pp. 233–246, 2016.

[5] L. Truong and T. Nguyen, "QoA4ML – A Framework for Supporting Contracts in Machine Learning Services," 2021 IEEE International Conference on Web Services (ICWS);, p. 11, 2021.

[6] M.-L. Nguyen, T. Phung, D.-H. Ly, and H.-L. Truong, "Holistic Explainability Requirements for End-to-End Machine Learning in IoT Cloud Systems," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, (Notre Dame, IN, USA), pp. 188–194, IEEE, Sept. 2021.

[7] S. K. Jagatheesaperumal, Q.-V. Pham, R. Ruby, Z. Yang, C. Xu, and Z. Zhang, "Explainable AI Over the Internet of Things (IoT): Overview, State-of-the-Art and Future Directions," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 2106–2136, 2022. Conference Name: IEEE Open Journal of the Communications Society.

[8] K. Razavi, M. Luthra, B. Koldehofe, M. Mühlhäuser, and L. Wang, "FA2: Fast, Accurate Autoscaling for Serving Deep Learning Inference with SLA Guarantees," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 146–159, May 2022. ISSN: 2642-7346.

[9] Y. Bai, L. Chen, M. Abdel-Mottaleb, and J. Xu, "Automated Ensemble for Deep Learning Inference on Edge Computing Platforms," *IEEE Internet of Things Journal*, vol. 9, pp. 4202–4213, Mar. 2022.

[10] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[11] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? What form? What from?," *The VLDB Journal*, vol. 26, pp. 881–906, Dec. 2017.

[12] Y. R. Wang and S. E. Madnick, "A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective,"

[13] M. Schmitz, "Survey on Data Quality and Provenance," Available at `https://dl.comp.nus.edu.sg/bitstream/handle/1900.100/11099/TR11_21_Martin_Schmitz.pdf`.

[14] A. Woodruff and M. Stonebraker, "Supporting fine-grained data lineage in a database visualization environment," in *Proceedings 13th International Conference on Data Engineering*, pp. 91–102, 1997.

[15] P. Buneman, A. Chapman, and J. Cheney, "Provenance management in curated databases," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, (New York, NY, USA), p. 539–550, Association for Computing Machinery, 2006.

[16] T. Heinis and G. Alonso, "Efficient lineage tracking for scientific workflows," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), p. 1007–1018, Association for Computing Machinery, 2008.

[17] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," *ACM Trans. Database Syst.*, vol. 25, p. 179–227, jun 2000.

[18] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," *The VLDB Journal — The International Journal on Very Large Data Bases*, vol. 12, pp. 41–58, 09 2001.

[19] D. Logothetis, S. De, and K. Yocum, "Scalable lineage capture for debugging disc analytics," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), Association for Computing Machinery, 2013.

[20] B. Glavic, *Big Data Provenance: Challenges and Implications for Benchmarking*, pp. 72–80. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[21] X. Rong-na, L. Hui, S. Guo-zhen, G. Yun-chuan, N. Ben, and S. Mang, "Provenance-based data flow control mechanism for internet of things," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 5, p. e3934, 2021.

[22] L. G. Azevedo, R. Souza, R. M. Thiago, E. Soares, and M. Moreno, "Experiencing ProvLake to Manage the Data Lineage of AI Workflows," in *Anais Estendidos do XVI Simpósio Brasileiro de Sistemas de Informação (Anais Estendidos do SBSI 2020)*, (Brasil), pp. 206–209, Sociedade Brasileira de Computação (SBC), Nov. 2020.

[23] F. T. Jaigirdar, C. Rudolph, G. Oliver, D. Watts, and C. Bain, "What Information is Required for Explainable AI? : A Provenance-based Research Agenda and Future Challenges," in *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*, (Atlanta, GA, USA), pp. 177–183, IEEE, Dec. 2020.

[24] A. Karkouch, H. Mousannif, H. Al Moatassime, and T. Noel, "Data quality in internet of things: A state-of-the-art survey," *Journal of Network and Computer Applications*, vol. 73, pp. 57–81, Sept. 2016.

[25] "Prometheus monitoring system and time series database." [Online]. Available: `https://prometheus.io`. Accessed: April 2, 2023.

[26] L. Moreau, P. Groth, J.-F. Baget, and et al., "PROV-DM: The PROV data model," 2013. [Online; accessed February 10, 2023].

[27] A. Kale, T. Nguyen, F. C. Harris, C. Li, J. Zhang, and X. Ma, "Provenance documentation to enable explainable and trustworthy AI: A literature review," *Data Intelligence*, vol. 5, pp. 139–162, Mar. 2023.

[28] D. Pina, L. Kunstmann, F. Bevilaqua, I. Siqueira, A. Lyra, D. De Oliveira, and M. Mattoso, "Capturing Provenance from Deep Learning Applications Using Keras-Prov and Colab: a Practical Approach," *Journal of Information and Data Management*, vol. 13, Dec. 2022.

[29] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The Open Provenance Model: An Overview," in *Provenance and Annotation of Data and Processes* (J. Freire, D. Koop, and L. Moreau, eds.), vol. 5272, pp. 323–326, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Series Title: Lecture Notes in Computer Science.

[30] "Ray: A distributed computing framework." `https://ray.io/`. Accessed: April 12, 2023.

[31] Ultralytics, "YOLOv5: You only look once (v5)." `https://github.com/ultralytics/yolov5`, 2020. Accessed: March 10, 2023.

[32] Y. Yang, "YOLOv8: You only look once (v8)." `https://github.com/WongKinYiu/yolov8`, 2022. Accessed: March 10, 2023.

[33] OpenCV Contributors, "OpenCV: Open source computer vision library." `https://opencv.org`, 2000. Accessed: April 12, 2023.

A. SQL for Querying Data Lineage of an Aggregation Function

```
1  SELECT EF.id,
2         EF.NAME,
3         EF.requestid,
4         PQ.value -> 'QoA' AS QoA,
5         M.NAME            AS model
6  FROM   ensemblefunction EF
7         INNER JOIN used U ON U.activityid = EF.id
8         INNER JOIN prediction PT ON PT.id = U.entityid
9         INNER JOIN wasgeneratedby WGB ON PT.id = WGB.entityid
10        INNER JOIN predict P  ON P.id = WGB.activityid
11        INNER JOIN wasgeneratedby WGBP ON P.id = WGBP.activityid
12        INNER JOIN predictionquality PQ ON PQ.id = WGBP.entityid
13        INNER JOIN wasassociatedwith WAW ON P.id = WAW.activityid
14        INNER JOIN model M ON M.id = WAW.agentid
15 WHERE  EF.NAME = 'max_aggregate'
16 UNION ALL
17 SELECT EF.id,
18        EF.NAME,
19        EF.requestid,
20        PQ.value -> 'QoA'AS QoA,
21        'aggregated'     AS model
22 FROM   ensemblefunction EF
23        INNER JOIN wasgeneratedby WGB ON EF.id = WGB.activityid
24        INNER JOIN predictionquality PQ ON PQ.id = WGB.entityid
25 WHERE  EF.NAME = 'max_aggregate'
26 ORDER  BY 1
```