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, 29 Jan 2023

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

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

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

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** 27 Nov 2017	**Pages** 70	**Language** English

Abstract

Abtract

Keywords keyword

urn https://aaltodoc.aalto.fi

Contents

1. Introduction

In recent years, Machine Learning (ML) has been applied extensively to many problems, including customer journey optimization in marketing [1], particle identification in Physics [2], and mental health problems prediction in healthcare [3]. With their proven performance and increasing number of ML applications , several organizations are seeking third-party ML service providers to improve their operations, thus leading to the creation of a new business model — ML as a Service (MLaaS). As with other kinds of service, MLaaS requires a contract which is usually called a service level agreement (SLA) that sets the expectations and describes the delivered service to preserve the benefits of both stakeholders. However, the current SLA has yet to fulfill its preservation job because ML-specific attributes, such as data quality, inference accuracy, and explainability, have created new challenges. For instance, in Internet of Things (IoT) applications, many issues of the ML models have their root causes in the quality of data, which is impacted by many factors such as erroneous measurement, environmental noise, and discrete observations [4]. Then, because of the relationship between data quality and inference accuracy, ML models can produce false predictions when data quality problems exist, causing ML providers to be penalized based on the SLA. Another issue is that in MLaaS, the customers only submit the data and does not cooperate with the ML provider in the prediction task. Thus, from their perspective, ML models are considered to be a black box, which is insufficient in human life-related applications, such as e-health and autonomous vehicles, where uncertain decisions cannot be tolerated.

Increasing model explainability by interpreting the inference result to the customer is one approach to resolve the challenges caused by the black box characteristics and the relationship between data quality and ML service performance [5]. However, explainability aspects of ML models have been researched mostly in the training task [4], making it unclear about the appropriate utilization and implementation of ML-specific attributes and constraints. As an approach to support ML-specific service contracts, Linh et al. [6] 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, current implementation only focuses on monitoring metrics of each individual base model. Consequently, because the relationship between base models are challenging to monitor and explain with the current state of QoA4ML, they remain a black box to both stakeholders. Such a level of explainability is not sufficient for some sophisticated applications, such as digital assistants, where a complex chain of models is employed. Additionally, with the increasing amount of research in autoscaler for the inference serving system [7] and automated ensemble [8], it demands a new technique to capture and visualize underlying inference graph to ensure dynamic and robust inference capabilities of ML solutions.

Recognizing above problem can be formalized as capturing data lineage, this thesis performs an analysis of data lineage monitoring and its implementation in edge environment. From that, the research proposes an approach to improve QoA4ML framework with data lineage, which can be combined with other techniques to improve the current MLaaS. This paper provides a prototype with QoA4ML as the foundation so that ML developer and provider can integrate it with their deployed service.

The rest of this paper is organized as follows. Section 2 discusses the background of the research, and section 3 explains data lineage implementation in the QoA4ML framework. Section 4 describes the experiment and its result, while section 5 concludes the research.

2. Background

In this section, the first part presents the background on ensemble ML and its explainability challenge. Then, in Section 2.2, data lineage will be discussed to explain how it has been utilized to increase the exaplainability of many systems. Lastly, Section 2.3 discusses one standard for data model of provenance which is employed in section 3.

2.1 Ensemble ML and its Explainability Challenges

Ensemble ML is a conventional technique that involves combining multiple models, which can reduce the variance of prediction, to improve the accuracy and robustness of predictions. This 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 has been shown to be highly effective in many applications including financial forecasting [9], image recognition [10], and natural language processing [11].

However, as the demand for more precise forecasts and advanced applications, such as AI assistants or autonomous vehicles, increases, it requires a complex dataflow graph, usually represented by a direct acyclic graph (DAG), comprised of multiple base models that are interconnected, ensembled, and orchestrated to handle the input data sequentially or in parallel [7]. For each prediction request, the whole DAGs 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 base models. Thus, when the performance of some, or even one, base models declines because of data quality problem, the quality of ensembled prediction will also decrease, potentially resulting in the violation of the SLA.

In addition, in recent years, other challenges have appeared from new techniques. For instance, in resource-constrained environments like the edge, ensemble selection rules can be applied so that only models that provide the best inference performance are utilized [8]. Although this approach can help resolve the challenges of high computation complexity, high resource occupation, and the moderate inference time of ensemble ML, it introduces uncertainty to the inference flow, making the data pipeline less transparent and explainable. Another technique that was proposed by Razavi et al. [7] can efficiently auto scale deep learning inference serving system to align with strict SLA on the end-to-end latency. Those two automated techniques are crucial components of a dynamic inference systems that is challenging for the current interpreting techniques to explain without capturing data lineage of the ML model.

2.2 Data Lineage

The terms lineage, provenance, and traceability can be used interchangeably to refer to the process of constructing a final product, whether it is a digital file or a physical item [12]. Regarding lineage for the digital file, Wang et al [13] was the first paper to discuss those issues, which helped formally define provenance as “data source tagging” and “intermediate source tagging” problems [14, p. 5]. From its original interpretation, provenance has been applied in many domains such as scientific databases [15] [16] [17] [18], data warehouses [19] [20], and recently big data platforms [21] [22], and IoT [23]. In different applications, the types of lineages, their generating techniques, and main issue that they can answer are diverse [12]. 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 [24] 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 merely represented by semantic relationships [25]. Consequently, while entire pipelines can be documented with provenance, the specific inner workings of individual models remain opaque,

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 [28]

which is not sufficient to explain the dynamicity of the serving system.

Additionally, provenance is often considered when assessing data quality tasks, e.g., evaluating integrity, trust, and accuracy of the data. By analyzing 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 [26]. Then, when combined with another monitoring service like Prometheus [27], ML provider can moderately explain the relationship between 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 [6].

2.3 PROV Family

Researchers have proposed various models, languages, and tools to facilitate the documentation of provenance, including those tailored for AI/ML models [29]. However, it is still crucial to have a standard reusable ontology to capture provenance in heterogeneous environments, such as the web, for further development of provenance documentation. 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 [28]. In those documents, the most significant one 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 can be 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 manner by *agent*. In the following paragraphs, the proper definition and application of those data models will be discussed

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 that describe their characteristics, such as their 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 entities that are responsible for carrying out activities (“wasAssociatedWith”), and when one activity is finished, an entity will be 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 the delegation relationship among them.

Section 3 will discuss the implementation of PROV-DM to capture data lineage of ensemble ML models.

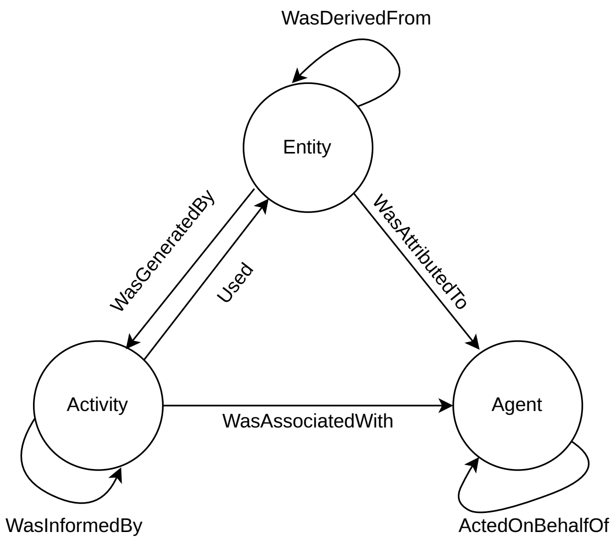

Figure 2.1. W3C PROV-DM structure

2.4 Related Work

Data lineage/provenance in AI systems: Data provenance is getting more attention when the demand for Explainable AI (XAI) increases. For instance, some researchers have proposed how provenance should be implemented: [30] introduced 'Six Ws' framework for provenance graph-based XAI; [31] combined abstraction and reasoning support offered by models with provenance graph which allows tracing the process of producing the current state.

3. Data Lineage for Ensemble ML Serving

3.1 PROV-DM Comparison With Other Approaches

One alternative to PROV-DM is the open provenance model (OPM) [32]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 which is enabled, facilitated, controlled, and affected by the *agent* [32, 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, and 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, there are some aspects of PROV-DM that are more beneficial than the OPM, and the following paragraphs discuss these advantages.

Firstly, while OPM was originally 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. [33] included new *referred* and *hadStore* relationships to represent data references in a heterogeneous database while Pina et al. [34] proposed a new domain-specific data model based on PROV-DM to represent training-specific data from deep learning experiments called DNNProv-Df.

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

Secondly, as OPM is a generic provenance model, its granularity level is lower than PROV-DM. Specifically, while OPM can only describe causal relationships, 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 not sufficient for explaining ensemble ML which requires a decent amount of data about all the base models.

Lastly, the higher level of adoption of PROV-DM compared to OPM is beneficial for the ML provider. As all stages of ML life cyle 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 [34] to fine tune 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 selection to represent the provenance data of ML applications. And by applying it, our solution can automatically capture provenance data at run time, send some of its data to monitoring service, and save it to the database for further data analysis and inference explanation.

3.2 Architecture

3.2.1 Overview and Design Principles

As the target of the research is data lineage in the production stage, there are some requirements that must be incorporated in the design principle of the library. Firstly, as ML serving requires alignment with the strict SLA about response time, the ML provider cannot afford a high runtime overhead during prediction. 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 code instrumentation. Moreover, the low instrumentation helps reduce time and resources to incorporate the latest updates to the model. Finally, the choice of the database for storing the data lineage needs to be scalable to support a large amount of writing requests while foster various kinds of provenance and data analysis. With those mentioned requirements, I implemented the library following these design principles:

- Lightweight: My 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 the crucial activities, such as predicting, preprocessing, and ensembling, in the specific order that is executed in the systems. Moreover, my approach handles the data lineage insertion and related relationship into the database in the background, requiring no attention of the user.

- Asynchronicity: As an ML serving system processes a huge number of requests concurrently in a strict SLA, an applicable solution manages the required time-consuming tasks to run asynchronous and not block the code execution. Then, my library utilizes the Python Threading module to run most of the tasks asynchronously. Although synchronous functions exist in my library, the overall performance with data lineage capturing is still guaranteed, and it is discussed 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 huge number of requests demands a scalable database that can handle the high write rate of capturing data lineage tasks. Thus, the choice for database is probably 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, I discuss the implementation in the following sections.

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, I create 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 verified with the database to determine if the model exists in our database. If not, it is recorded and give an `id` to the class to identify individual model. This automatic model capturing can help with investigating the relationship between model parameters and the predicting performance.

Secondly, the methods in different class 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 captures the necessary input and output of the task, which are asynchronously inserted into the database. Additionally, the decorator can calculates 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.

Lastly, an entity, which is the targeted product of provenance capturing, is used to abstract the input and output of activity executed in the system. Corresponding with four activity types, there are four entity classes: `Data`, `DataQualityReport`, `Prediction`, and `PredictionQuality`. When the client submits their data to ML provider, it is first captured with the function `captureInputData` along with the unique `requestId` to identify the individual request in the database. Then, the input and output of the functions that are decorated with `capture` are recorded and returned to be utilized in later functions. This process of transforming the output into an equivalent entity class is handled by the decorator, reducing the need for code modifications. However, because the proposed entity classes are an abstraction for only the input and output of targeted activity, it requires the variable to be taken outside the class to be employed by other functions.

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, which must be aligned with our design principle listed before, to store those data. Thus, my choice is a relational database and specifically PostgreSQL for two reasons.

Firstly, relational databases offer a high level of flexibility in managing and organizing data. As the data is efficiently stored in a structured manner that has 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 my proposed schema is only sufficient in some certain scenario. In other cases, it may requires the modification of database schema to capture the whole data lineage. Additionally, relational databases foster complex queries that provide flexibility in data manipulation and analysis. This 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 highly scalable, making them suitable for capturing the huge 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, my choice of PostgreSQL, an open-source database, supports a high level of performance customizability which comes 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

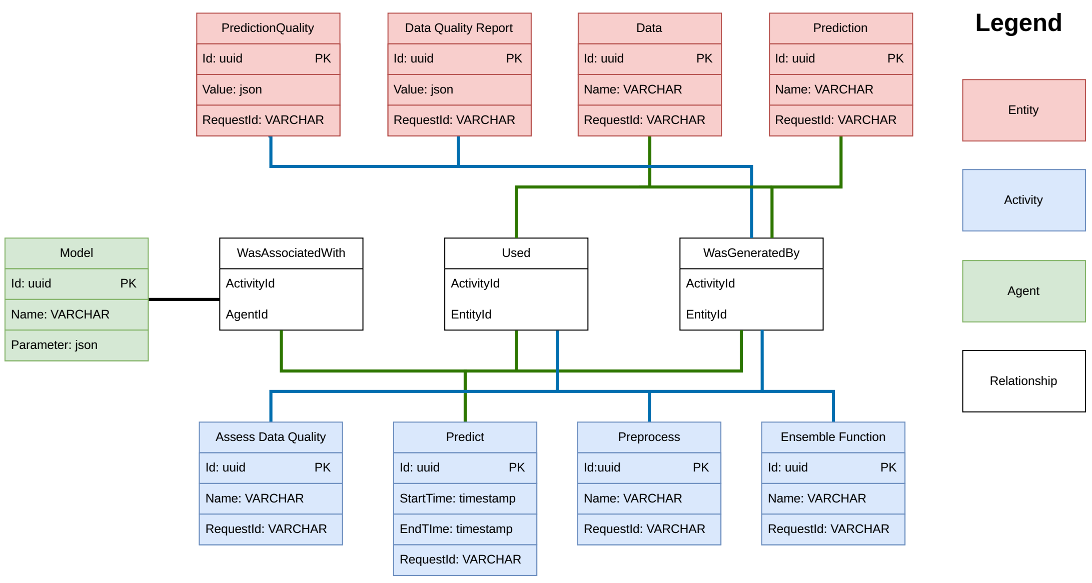

Figure 3.1. Database Schema

In relational databases, it requires a database schema that defines the organization and arrangement of data. Thus, as my solution employs PROV-DM data models, the tables in my proposed schema, which can be seen in Figure 3.1, belongs 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 there are different entity and activity tables having 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 individual false prediction, (iii) efficiently understand the impact of each base model to 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. One of those applications will be experimented in Section 4.

4. Experiment and Results

In this section, I perform an experiment to demonstrate the low computational overhead when integrating the library into an example ML serving system. Then, from the data lineage that is generated in the first experiment, a simple pricing model which is based on resource usage and accuracy is applied as an example of application that is simpler to executed with my solution. All experiments are performed utilizing our proposed library.

4.1 Experiment Settings

I describe the experiment setup, including the hardware, an example serving system framework, an ensemble ML model with a one sample dataset, and PostgreSQL database with its Python adapter in the following paragraphs.

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, stable power supply, and appropriate delay between runs to ensure reliability during the experiment.

Serving system framework. Ray [35] 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, which makes it quickly to implement a 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. My choice of ML application for the experiment is object detection as it is extensively utilized with various pre-trained models. Then, Yolov5 [36] and Yolov8 [37] are selected as they are both open-source project developed by Ultralytics, which is known for the simplicity in terms of implementation and setup. Because both models have pre-trained version so for low computing power, Yolov5s and Yolov8n version are elected. After both models have generated their prediction, they will be ensembled with a simple max aggreation 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), and in these experiments, Google Open Image Dataset V1 is selected. Then, from 600 classes in the dataset, I chose only five and each class has 100 images that are put into five different folders: cat, dog, person, car, and bicycle.

PostgreSQL and Python adapter. PostgreSQL version 15.2 with default setting is locally hosted using Docker container. Then, I utilize Psycopg3 version 3.1 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, I employed the same deployment of example ensemble model *with* and *without* data lineage captured for handling request from one client. The experiment uses a fixed seed value of 1234 to ensure consistent random image selection. A random class folder will be selected using the seed, and then a random image within that folder will be sent to Ray cluster through HTTP request. The response time for each request will be recorded, which is calculated by the time taken when the client makes the request until receiving the response. In total, 1000 requests are sent consecutively.

The first 300 requests will be ignored to mitigate any potential warm-up effects or initial resource consumption variances and guarantee reliable results. With the remaining 700 requests, the mean and p99 response time will be calculated to analyze the effect capturing data lineage to 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.

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

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. This is mostly because the resource 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 mean 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.

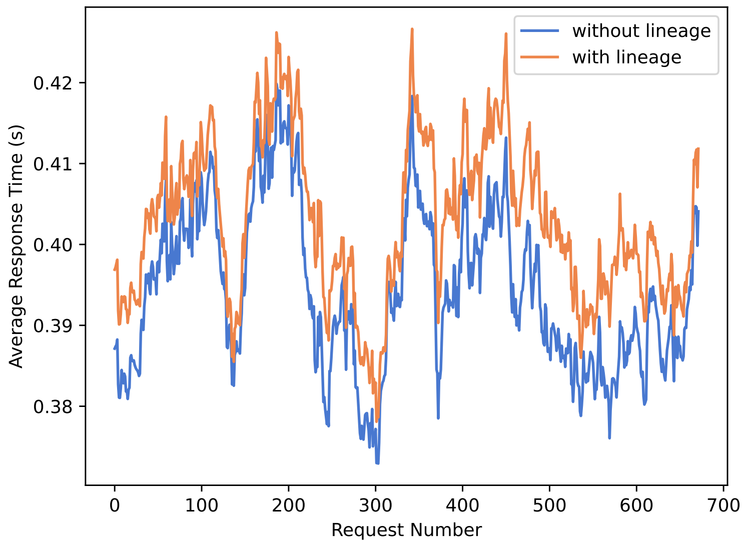

Figure 4.1. Mean Response Time Over 700 Request with Window Size of 30

4.2.2 Pricing Model based on Resource Usage and Accuracy

Traditional ML pricing models frequently ignore other crucial considerations such as resource utilization, data quality, and prediction performance and only focus on the number of API calls or the complexity of the application. In this experiment, we suggest a novel pricing model that considers these extra facots in order to offer a reasonable and effective pricing structure for ML providers and their clients. The experiment involves data lineage generated from the previous experiment to be utilized as the input for two pricing mode, which are my proposed model and the conventional model:

- Proposed Pricing Model: In this model, the cost for the customer will be calculated based on the resource usage of their request, such as CPU or memory usage, the quality of the input data used for prediction, and the prediction performance achieved by the ML model. A database will be used to store and track these metrics for each customer request.

- Traditional Pricing Model: In this model, the cost for the customer will be calculated solely based on the number of API calls made, without considering resource usage, data quality, or prediction performance.

The experiment will involve randomly selecting a subset of customer requests from the dataset and applying both pricing models to calculate the cost for each request.

5. Conclusion

Capturing data lineage in ML systems is crucial as the number of ML application along with MLaaS will continue to grow, which requires a concise monitoring solution for model explanation to preserve the stakeholders' benefit and comply with the incoming AI acts. In this research, I address the challenges of individual model monitoring by applying PROV-DM data model to capture the data lineage of an example end-to-end ML system. I have implemented a lightweight library that can be easily integrated into the existing serving system without too many code modifications. Moreover, the choice for database with corresponding schema is proposed which supports both flexibility and moderate scalability. Then, the experiment is performed to show the effect of capturing data lineage to ML system's performance and one simple pricing model that is based on resource usage and accuracy is applied to demonstrate an example application that is enabled by data lineage.

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

Bibliography

[1] A. Terragni and M. Hassani, “Optimizing customer journey using process mining and sequence-aware recommendation,” in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC ’19, (New York, NY, USA), p. 57–65, Association for Computing Machinery, 2019.

[2] D. Derkach, M. Hushchyn, T. Likhomanenko, A. Rogozhnikov, N. Kazeev, V. Chekalina, R. Neychev, S. Kirillov, F. Ratnikov, and on behalf of the LHCb collaboration, “Machine-learning-based global particle-identification algorithms at the lhcb experiment,” *Journal of Physics: Conference Series*, vol. 1085, p. 042038, sep 2018.

[3] A. E. Tate, R. C. McCabe, H. Larsson, S. Lundström, P. Lichtenstein, and R. Kuja-Halkola, “Predicting mental health problems in adolescence using machine learning techniques,” *PLOS ONE*, vol. 15, pp. 1–13, 04 2020.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] M. Schmitz, "Survey on Data Quality and Provenance,"

[15] 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.

[16] 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.

[17] C. Ré and D. Suciu, "Approximate lineage for probabilistic databases," *Proc. VLDB Endow.*, vol. 1, p. 797–808, aug 2008.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] B. Glavic, *Big Data Provenance: Challenges and Implications for Benchmarking*, vol. 8163, pp. 72–80. 01 2014.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] L. Moreau and P. Missier, “Prov-dm: The prov data model,”

[29] 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.

[30] 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.

[31] O. Reynolds, A. García-Domínguez, and N. Bencomo, “Automated provenance graphs for models@run.time,” in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, (Virtual Event Canada), pp. 1–10, ACM, Oct. 2020.

[32] 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.

[33] R. Souza, L. Azevedo, R. Thiago, E. Soares, M. Nery, M. A. S. Netto, E. Vital, R. Cerqueira, P. Valduriez, and M. Mattoso, “Efficient Runtime Capture of Multiworkflow Data Using Provenance,” in *2019 15th International Conference on eScience (eScience)*, (San Diego, CA, USA), pp. 359–368, IEEE, Sept. 2019.

[34] 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.

[35] “Ray: A distributed computing framework.” `https://ray.io/`. Accessed: April 12, 2023.

[36] Ultralytics, “YOLOv5: You only look once (v5).” `https://github.com/ultralytics/yolov5`, 2020.

[37] Y. Yang, “YOLOv8: You only look once (v8).” `https://github.com/WongKinYiu/yolov8`, 2022.