PROVENANCE DATA AS A FIRST-CLASS CITIZEN FOR DEEP LEARNING WORKFLOW ANALYSES

Débora Barbosa Pina

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Rio de Janeiro

Março de 2025

PROVENANCE DATA AS A FIRST-CLASS CITIZEN FOR DEEP LEARNING WORKFLOW ANALYSES

Débora Barbosa Pina

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Aprovada por: Prof. Marta Lima de Queirós Mattoso
Prof. Daniel Cardoso Moraes de Oliveira
Prof. Adriane Chapman
Prof. Aline Marins Paes Carvalho
Prof. Altigran Soares da Silva
Prof. Claudio Miceli de Farias
Prof. Fábio André Machado Porto

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2025

Barbosa Pina, Débora

Provenance data as a first-class citizen for deep learning workflow analyses/Débora Barbosa Pina. – Rio de Janeiro: UFRJ/COPPE, 2025.

XV, 144 p.: il.; 29, 7cm.

Orientadores: Marta Lima de Queirós Mattoso

Daniel Cardoso Moraes de Oliveira

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2025.

Referências Bibliográficas: p. 124 – 144.

1. Deep Learning. 2. Provenance. 3. W3C PROV. I. Lima de Queirós Mattoso, Marta *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha família.

Agradecimentos

Agradeço, primeiramente a Deus, pela força e sabedoria para concluir esta jornada. À minha família, meus pais e minha irmã, meu alicerce inabalável, sou imensamente grata por todo o apoio, amor e compreensão que recebi ao longo desses anos. Agradeço a cada um de vocês por sempre estarem ao meu lado, nos momentos de alegria e nos de dificuldade, oferecendo força, encorajamento e um carinho que nunca falhou. O sacrifício, a paciência e a confiança que vocês depositaram em mim foram essenciais para que eu chegasse até aqui.

Expresso minha gratidão à minha orientadora, Prof. Marta Mattoso, com quem tive o privilégio de colaborar desde a graduação. Sua orientação cuidadosa, sua dedicação inabalável e sua busca incansável pela excelência sempre me inspiraram. Mais do que uma referência acadêmica, a Profa. Marta é um exemplo de liderança e competência, especialmente para mulheres na ciência. Ver seu reconhecimento e impacto na comunidade científica me motiva a seguir em frente e a acreditar no meu próprio potencial. Sou profundamente grata por todas as oportunidades de aprendizado e crescimento que tive sob sua orientação, bem como pela confiança que sempre depositou em mim.

Ao Prof. Daniel de Oliveira, meu coorientador, cuja orientação foi essencial para o desenvolvimento desta tese, sou imensamente grata. Agradeço não apenas pelos ensinamentos e pelas críticas construtivas, mas também pelo apoio constante e pela confiança no meu trabalho. Sua paciência e a leveza com que lida com inseguranças e nervosismos tornaram os desafios mais fáceis de superar. Seu profissionalismo, aliado à forma cuidadosa com que orienta, sempre inspirou confiança em mim.

À Prof. Adriane Chapman, que me acolheu durante o doutorado sanduíche em Southampton. Sua receptividade, orientação e apoio foram inestimáveis nesse período, tornando essa experiência enriquecedora tanto academicamente quanto pessoalmente.

À minha querida amiga Andréa Doreste, a quem sou eternamente grata, quero expressar toda a minha admiração e gratidão. Desde a graduação e mestrado, temos compartilhado o desafio da pesquisa científica, e, mesmo agora no doutorado, com a distância que nos separa, continua sendo uma presença fundamental na minha vida. Sua amizade é um

verdadeiro presente. Nos momentos de insegurança e dúvida, suas palavras de apoio e compreensão sempre foram um alicerce, e em cada vitória, mesmo à distância, sua alegria e celebração tornaram tudo ainda mais especial.

Durante o doutorado, tive a sorte de contar com colegas que não só compartilharam comigo os desafios dessa jornada, mas também tornaram o caminho mais inspirador e leve. Agradeço aos colegas Liliane Kunstmann, Lyncoln de Oliveira, e Filipe Silva, pelas valiosas colaborações e por serem fontes constantes de inspiração para seguir em frente. A convivência com vocês fez o dia a dia de trabalho mais prazeroso e, sem dúvida, foi essencial para minha perseverança. Sou profundamente grata por todas as conversas, pelo apoio incondicional e pelo companheirismo ao longo de todos esses anos.

Ao Vítor Silva, que me orientou na graduação e continua sendo uma presença constante em minha trajetória acadêmica. Agradeço pela disposição em sempre oferecer seu apoio, seja para *brainstorm*, desenvolvimento de ideias, ou até mesmo para orientar meus passos na oratória. Sua generosidade, paciência e disponibilidade fizeram toda a diferença, e sou muito grata por ter contado com sua orientação.

Aos membros da administração do PESC e do NACAD, Gutierrez da Costa, Mara Prata e Ricardo Cezar, pelo suporte e dedicação que tornam o dia a dia acadêmico mais fluido. Ao Professor Álvaro Coutinho, pelo apoio contínuo por meio de projetos e pela viabilização de estudos de caso reais, que foram fundamentais para este trabalho.

Aos membros da banca, Professores Adriane Chapman, Aline Paes, Altigran Soares, Claudio Miceli e Fabio Porto, pelo tempo dedicado a avaliar esta tese e pelas contribuições valiosas. Por fim, às agências de fomento CAPES, CNPq e FAPERJ, pelo suporte financeiro que possibilitou a realização deste trabalho. Além disso, agradeço ao INRIA, LNCC, NACAD, e UFF pela disponibilização de recursos computacionais.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

DADOS DE PROVENIÊNCIA COMO CIDADÃO DE PRIMEIRA CLASSE PARA ANÁLISES DE WORKFLOWS EM APRENDIZADO PROFUNDO

Débora Barbosa Pina

Março/2025

Orientadores: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Programa: Engenharia de Sistemas e Computação

Workflows de aprendizado profundo (AP) consistem em várias etapas conectadas e repetitivas, incluindo preparação de dados, treinamento, avaliação e implantação de modelos. Cada etapa envolve decisões que impactam o desempenho e a aplicabilidade do modelo final. É essencial rastrear as etapas do *workflow* que geraram o modelo de AP, para oferecer qualidade e confiabilidade ao modelo. No entanto, as soluções de rastreabilidade existentes frequentemente não oferecem o encadeamento do *workflow* de AP, dependem de formatos proprietários ou não geram documentos de proveniência que acompanhem os modelos em produção. Essas limitações comprometem os benefícios da rastreabilidade. Apresentamos o DLProv, um conjunto de serviços de proveniência para apoiar a rastreabilidade em *workflows* de AP. DLProv apoia consultas em SQL durante o treinamento e gera documentos de proveniência que registram a preparação de dados, o treinamento e a avaliação do modelo. Esses documentos seguem o padrão W3C PROV, garantindo interoperabilidade. O DLProv possui uma arquitetura independente de *framework* de AP, mas também inclui instâncias especializadas para Keras e Redes Neurais Informadas por Física (PINNs). Avaliamos o DLProv em estudos de caso, desde arquiteturas consolidadas de AP até PINNs, mostrando sua capacidade de capturar e gerenciar dados de proveniência, assegurando rastreabilidade em conformidade com o W3C PROV. Os experimentos também analisam o uso do documento de proveniência gerado, permitindo identificar problemas no modelo de AP implantado. Nossos experimentos também mostraram que a integração dos serviços do DLProv não comprometem o tempo de execução do workflow.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PROVENANCE DATA AS A FIRST-CLASS CITIZEN FOR DEEP LEARNING WORKFLOW ANALYSES

Débora Barbosa Pina

March/2025

Advisors: Marta Lima de Queirós Mattoso
Daniel Cardoso Moraes de Oliveira

Department: Systems Engineering and Computer Science

Deep Learning (DL) workflows consist of multiple connected and repetitive steps, including data preparation, model training, evaluation, and deployment. Each step involves decisions that impact the final model's performance and applicability. It is important to trace the workflow steps that generated the DL model to provide quality and trust to the model. However, existing traceability solutions often do not provide DL workflow traceability, rely on proprietary formats, or fail to generate provenance documents that can accompany models into production. These limitations compromise the benefits of traceability. We introduce DLProv, a suite of provenance services designed to ensure traceability in DL workflows. DLProv supports SQL-based querying during training and generates provenance documents that capture data preparation, model training, and evaluation. These documents comply with the W3C PROV standard, ensuring interoperability. DLProv features a framework-agnostic architecture, allowing integration with different DL frameworks, and includes specialized instances for Keras and Physics-Informed Neural Networks (PINNs). We evaluated DLProv across multiple case studies, ranging from well-established DL architectures to PINNs, showing its ability to capture and manage provenance data while ensuring traceability in compliance with W3C PROV. Our experiments also highlighted the use of the provenance document generated after the model's deployment, enabling the identification of potential issues in the DL model. In addition, our evaluation showed that integrating DLProv services does not compromise workflow execution time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Learning (DL) is a subset of Machine Learning (ML) that supports decision-making processes by focusing on training computational models composed of multiple layers of nonlinear processing units (GOODFELLOW *et al.*, 2016; SCHMIDHUBER, 2015). These layers enable the learning of hierarchical data representations, which generate a DL model (LECUN *et al.*, 2015). The hierarchical learning capabilities depend on leveraging large, preprocessed datasets (DENG *et al.*, 2009). Once trained, a DL model can identify patterns in these datasets and others. As a result, DL has become a transformative technology, advancing fields such as Natural Language Processing (NLP), Computer Vision (CV), and Speech Recognition (SR) (CHAI *et al.*, 2021).

DL models are generated by leveraging datasets to be trained and validated by a Deep Neural Network (DNN) architecture (GOODFELLOW *et al.*, 2016). A DL model generation typically involves workflow steps such as data preprocessing, model training, hyperparameter tuning, and model validation (BOEHM *et al.*, 2019; CHAI *et al.*, 2023), all producing artifacts and metadata contributing to the model's final form ORMENISAN *et al.* (2020); SCHELTER *et al.* (2017); SCHLEGEL and SATTLER (2023a). Due to the variety of alternative configurations for each workflow step, several candidate DL models are usually generated (IDOWU *et al.*, 2024). Deciding which configurations to pursue in model development relies on analyzing the processes that led to the generated DL models.

Three key types of data support informed decision-making: data isolated from the DL model, model configuration, and model metrics. Data refers to the input examples used during the training, validation, and testing of DL models. These examples may consist of raw or preprocessed instances, and the way they are prepared can influence the model's performance and generalization capabilities (CHAPMAN *et al.*, 2020; POLYZOTIS *et al.*, 2018). Model configuration encompasses all factors influencing model generation, including hyperparameter configurations, DL architecture definitions, and preprocessing

decisions (BERGSTRA and BENGIO, 2012). Each combination of configuration values represents a distinct setup used to train a DL model with a certain quality. Finding the best configuration is a complex challenge due to the extensive space of candidate models to be evaluated. This process, known as *Model Selection*, involves training and assessing multiple models with different configurations to identify the most suitable one for future deployment (BRUNTON and KUTZ, 2019; SCHLEGEL and SATTLER, 2023a; SHALEV-SHWARTZ and BEN-DAVID, 2014). Model metrics, on the other hand, provide quantitative measures of a trained model's performance, such as accuracy, loss, and precision. These metrics guide the final selection of the most suitable DL model for deployment by enabling comparisons between candidate models.

After being selected for deployment, a DL model transitions into production, becoming part of real-world systems and applications. As discussed by SOUZA *et al.* (2024), the complexity of DL models with their application in critical decision-making requires trust in model predictions. Provenance tracking in DL workflows has emerged as an essential support for this trust and transparency (FERREIRA DA SILVA *et al.*, 2024). Providing provenance tracking for DL workflows requires capturing and relating metadata from all the workflow steps (PINA *et al.*, 2023, 2024, 2025; SOUZA *et al.*, 2022). Despite existing initiatives in the literature, tracking provenance relationships in DL workflows is an open, yet important, problem (FERREIRA DA SILVA *et al.*, 2024; LEO *et al.*, 2024).

Consider an illustrative scenario involving Alice, a data scientist working on a DL project to transcribe handwritten documents into digital text. During the model development stage, Alice experiments with various preprocessing configurations, such as binarizing images and applying noise reduction techniques. She also explores different DL architectures, different values for the learning rate, and optimizers. Each combination of these configurations represents a *trial*, leading Alice to evaluate many distinct model configurations. Now, imagine that Alice is using MLflow[1] (CHEN *et al.*, 2020; ZAHARIA *et al.*, 2018) to manage this DL project. Each experiment is tracked with metadata related to datasets, model configurations, and model metrics, as shown in Figure 1.1.

Now, consider a scenario where Alice discovers that certain handwritten words, especially those with complex or uncommon shapes, are consistently misrecognized. To troubleshoot this, she would ideally need to query the system for a sequence of details: identifying the dataset version used, the preprocessing steps applied, and the specific model configurations that may have led to the misrecognition. However, with MLflow, it would be difficult to perform a relational query, such as *"Which dataset and preprocessing operations were used to train DL models that exhibited high error rates?"*.

[1] `https://mlflow.org`

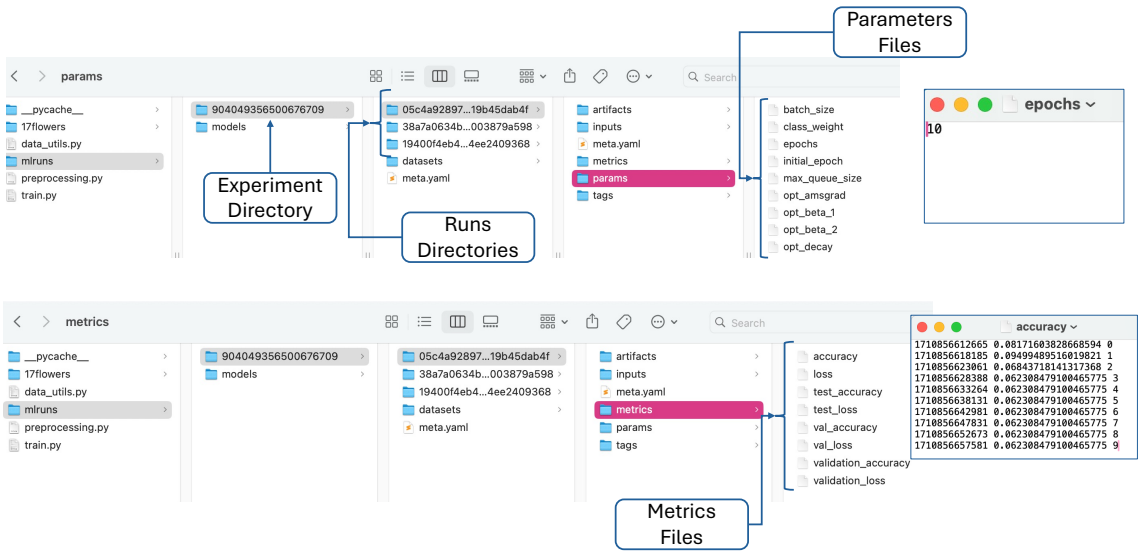


Figure 1.1: MLflow files for a particular experiment.

Although MLflow allows Alice to filter individual runs by parameters such as the learning rate by querying, like *"Which training runs used a particular learning rate?"*, more complex queries become challenging. MLflow lacks native support for tracing back through each workflow step to analyze dependencies and relationships among artifacts, such as datasets, preprocessing steps, and resulting models. As a result, Alice must manually reconstruct the metadata records, which is labor-intensive and susceptible to errors. This challenge becomes even more pronounced in complex workflows that involve numerous sequential steps and interconnected artifacts, further complicating the accuracy and efficiency of the reconstruction process.

These metadata analyses can operate at an entity level, akin to analytical queries, or at a relationship level, operating over provenance traces (PINA *et al.*, 2024). Entity-level queries focus on analyzing metadata associated with a dataset or trained model (SCHELTER *et al.*, 2017). For instance, analytical queries can involve aggregating and filtering model configuration values and model metrics of a DL workflow execution to gain insights into the model training process. On the other hand, relationship-level queries inspect relationships between entities, activities, or agents in the DL workflow provenance trace (SOUZA *et al.*, 2022). They aim to trace the lineage of how entities were used or generated throughout the activities in the workflow.

Current DL frameworks often employ proprietary traceability representations, creating an ecosystem that makes transparency difficult (MORA-CANTALLOPS *et al.*, 2021). Existing solutions for monitoring and analyzing DL models typically concentrate on providing analyses through metadata management without traceability. While some solutions,

like Weights & Biases[2], have some support for traceability, they lack the representation of typical relationships, requiring additional programming effort to construct and follow them. This limitation restricts the ability to trace derivation paths across the DL workflow steps. Moreover, we consider that provenance capture for traceability should not be tightly coupled to the DL framework (PINA *et al.*, 2024). With the diverse landscape of DL frameworks and execution environments, DL scientists should be free to use different frameworks independently for each DL workflow step.

Unlike traditional software systems, DL workflows often run in environments ranging from cloud-based platforms to high-performance computing (HPC) clusters (SERGEEV and BALSO, 2018), with each environment imposing its requirements and limitations. This scenario contributes to the challenge of DL workflow provenance tracking and calls for a solution that flows along different frameworks and platforms to generate a DL provenance graph. Therefore, the research questions this thesis addresses are:

- **RQ1: Traceability:** How can we capture and integrate data derivation paths in DL workflows to support traceability from data preprocessing to model deployment?
- **RQ2: Provenance Representation:** How can provenance data be represented to foster interoperability while avoiding ad-hoc representations?
- **RQ3: Arbitrary Execution Environment:** What strategies can be implemented to enable provenance capture and analysis, allowing scientists the freedom to use arbitrary frameworks?
- **RQ4: Generation of an Independent Provenance Document:** What methods can be developed to create a provenance document that remains independent of the original capture solution?

The lack of traceability prevents typical relationships that define derivation paths, hindering relationship queries and compromising reproducibility analyses. Additionally, the lack of a standardized provenance representation or adherence to established recommendations leads to provenance data being stored in isolation, making it difficult to compare across systems and limiting its contribution to reproducibility across different projects (DA SILVA *et al.*, 2023b; KHAN *et al.*, 2019). This also impacts transparency, as provenance records become fragmented and less accessible. The lack of support for arbitrary execution environments reduces the flexibility of a solution, making it difficult to operate across the diverse landscape of DL frameworks and execution environments (RUPPRECHT *et al.*, 2020; TSAY *et al.*, 2018). This, in turn, prevents the ability to reproduce DL workflows in different settings. The lack of an independent provenance document further restricts the ability to analyze and verify DL workflows after deployment. Without

[2] `https://wandb.ai/site`

a self-contained provenance record accompanying the DL model, scientists must rely on the original environment where provenance was captured, limiting post-deployment analyses and increasing dependence on specific tools or infrastructure. From this foundation, the research goals are defined as follows:

- To develop an independent provenance service that integrates provenance data across various stages of DL workflows, including DL model configuration, DL model training and evaluation, and dataset transformation.
- To establish a standardized provenance data representation that enables compatibility across diverse ML and HPC frameworks, promoting consistency in how provenance is captured, stored, and analyzed.
- To design a solution that accommodates the computational requirements of both small-scale and HPC applications and can be integrated within scientists' preferred frameworks and libraries, focusing on the high-performance needs of specialized DL applications, such as Physics-Informed Neural Networks (PINNs).
- To create a methodology for generating an independent provenance document that can be used beyond the original provenance data capture context, ensuring that it serves as a standalone artifact for analysis.

The challenges are developing an approach that is *(i)* simultaneously representative and extensible; *(ii)* compliant with provenance query standards; *(iii)* equipped with resources for interoperability across steps of the life cycle; *(iv)* platform-independent; and *(v)* characterized by low overhead. By addressing these challenges, this thesis has two main contributions:

- Enhancing analytical capabilities during the development and selection of DL models.
- Integrating aspects of trust, transparency, and quality into models in production.

We have conducted a literature review, and to the best of our knowledge, we found no previously proposed solution that offers traceability support that integrates the DL development steps and delivers a provenance graph document to follow the DL model in production workflows. The hypothesis of this thesis posits that by providing provenance data support, encompassing metadata and relationships, as a service for DL frameworks, data scientists can make more informed decisions during model development and selection while helping with the trust and reproducibility of deployed DL models.

Provenance data is a natural solution for the traceability of model configurations, model metrics, and other artifacts used and generated in DL workflows. Provenance data can be

compliant with the recommendation of the World Wide Web Consortium (W3C), specifically the W3C PROV model (BELHAJJAME *et al.*, 2013; MOREAU and GROTH, 2013), preventing the main problems of *ad-hoc* representation, and can be captured and queried using services that can be independent of DL frameworks. Our work operates under the premise that a provenance solution, working independently of any single ML framework or library and capable of scaling from local to HPC environments, is important to support provenance data as a first-class citizen that contributes with DL workflow analyses to trust, transparency, and quality of DL models. Provenance is also well known for its support of reproducibility (DAVIDSON and FREIRE, 2008).

To address the lack of traceability of the DL workflow artifacts during the DL workflow, we propose a specific strategy to capture, store, and query provenance data and, later, integrate the provenance data captured in different steps of the DL workflow. Integrating DL provenance data from all steps of the DL workflow is not a trivial task. Data scientists may choose different tools to execute each workflow step according to their needs and requirements.

To address the ad-hoc representation problem, we propose a specific provenance schema representing DL workflow artifacts and integration with the preprocessing provenance traces. The provenance schema aims to provide a general representation of said DL artifacts, following the W3C PROV recommendation (MOREAU and GROTH, 2013) for representing provenance data. By adopting the W3C PROV-DM recommendation, this thesis aims to provide a public, extensible data representation adopted in several application domains, facilitating interoperability, reproducibility, and comparison among results.

To address the limited support for arbitrary execution frameworks, we propose a solution involving the implementation of a service of provenance capture using instrumentation calls. These calls can be integrated into scripts and DL frameworks, regardless of the programming language. To facilitate the instrumentation, interfaces for automatically instrumenting DL training scripts can be used. Data scientists can then upload their scripts, select the data to capture, and align them with script variables.

To address the need for an independent provenance document, we propose a solution that enables the export of provenance graphs in multiple formats, such as JSON and PROV-N, based on the proposed provenance representation. This solution ensures that the provenance graph can accompany the DL model in production environments. This provenance generation offers a level of autonomy, as certain analyses can be performed directly on the provenance graph after the DL model is deployed, without the need to return to the solution that captured and stored the provenance during the DL model training and the analysis of candidate models.

Therefore, we have developed DLProv, a suite of provenance services designed to support DL workflow analysis through a provenance data-centric approach. DLProv is built on a W3C PROV-compliant provenance representation (PINA *et al.*, 2021, 2023, 2024), tackling challenges *(i)* and *(ii)*. It generates traceability documents as provenance graphs, integrating the traces of DL models throughout their life cycle. DLProv supports the model generation workflow, enabling scientists to assess model performance and data quality, thereby intertwining the DL model with data. Its conceptual model facilitates integration with other W3C PROV-based provenance representations that capture complementary steps of the DL workflow (PINA *et al.*, 2023), addressing challenge *(iii)*. By leveraging instrumentation, DLProv captures provenance data independently of the DL framework used (PINA *et al.*, 2022; SILVA *et al.*, 2021b), aligning with challenge *(iv)*. As the DL workflow executes, provenance data is asynchronously stored in a Database Management System (DBMS), enabling flexible representation of entities and relationships and supporting independent, query-based analysis. In addition, DBMSs naturally support concurrency control, fault tolerance, recovery mechanisms, and transactions, making them a robust foundation for provenance data persistency (GARCIA-MOLINA *et al.*, 2008). This approach ensures low overhead (PINA *et al.*, 2021; SILVA *et al.*, 2021b), addressing challenge *(v)*. DLProv is publicly available at `https://github.com/dbpina/dlprov`.

This thesis is organized into five chapters in addition to this introduction. Chapter 2 presents background information on DL workflows and provenance data, while Chapter 3 reviews relevant related work. Chapter 4 presents DLProv, our proposed approach, and Chapter 5 details the experimental evaluation within DL workflows. Finally, Chapter 6 provides the concluding remarks and discusses future directions for this research.

Chapter 2

Background: Challenges in Traceability of Deep Learning Workflows

This chapter introduces important concepts of this thesis. It starts with basic concepts of scientific workflows that can be applied to DL workflows. Following, we introduce concepts involved in the DL workflow, providing an overview of the steps and elucidating how model selection is carried out. Then, we highlight the significance of human involvement in DL workflows, exploring its pivotal role. Finally, the concepts of provenance data and its importance in ensuring traceability within DL workflows

2.1 Basic Concepts

The concepts introduced in this thesis are inspired by the work of SILVA *et al.* (2020) on scientific workflows and are applicable to the DL workflow. Considering that a *dataset* presents any set of *data elements* to be consumed or produced by a data transformation, it is assumed that each dataset has pre-defined attributes that are present in each data element (Definition 2.1.1). In the *dataflow* scenario, a *data transformation* performs any processing based on procedures from an algorithm or computational model, consuming data from one or more datasets as input and producing one or more datasets as output (Definition 2.1.2). Two data transformations establish a *data derivation trace* concerning a dataset when the data elements are generated by one data transformation and consumed by another (Definition 2.1.3). With these foundation ideas, a dataflow is the composition of data transformations (Definition 2.1.4). A dataflow can be described by a *Directed Acyclic Graph* (DAG), where the nodes denote the data transformations and the edges denote the flow of datasets that are "input to" and "output of" data transformations.

Definition 2.1.1 (Dataset, Data Elements, and Data Values)**.** *A dataset DS is composed of data elements, i.e. $DS = e_1, ..., e_m$. Each data element $e_i, 1 \leq i \leq m$, is composed of data values, i.e. $e_i = v_1, ..., v_n$.*

Definition 2.1.2 (Data Transformation)**.** *A data transformation is characterized by the consumption of one or more input datasets I_{DS} and the production of one or more output datasets O_{DS}. A data transformation is represented by DT, where $O_{DS} \leftarrow DT(I_{DS})$.*

Definition 2.1.3 (Data Derivation Trace)**.** *Let DT_α and DT_β be data transformations, and let $E \subset DS$ be a set of data elements produced in an output dataset DS generated by DT_α. If DT_β consumes the elements E, then DS is also an input dataset of DT_β. This defines a data derivation trace between $DT\alpha$ and DT_β through E, represented as $\varphi = (E, DT_\alpha, DT_\beta)$.*

Definition 2.1.4 (Dataflow)**.** *A dataflow $Df = (T, S, \Phi)$, is a triple composed of a set of data transformations T, a set of datasets S consumed and produced the data transformations in T, and Φ is the data derivation traces between the data transformations in T.*

These concepts are illustrated in Figure 2.1. It presents two chained data transformations, DT_1, and DT_2, with a data derivation trace connecting them through data elements of the dataset I_{DS2}, which is both an output dataset for DT_1 and an input dataset for DT_2.

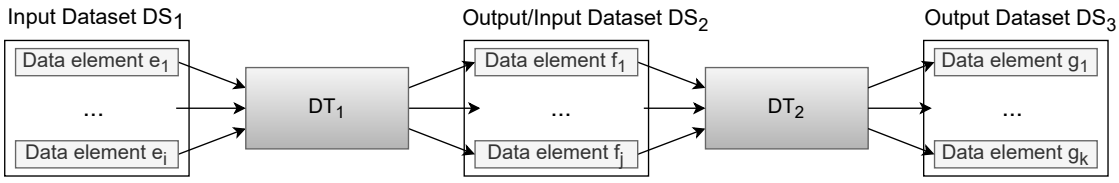


Figure 2.1: Example of a dataflow with data transformations DT_1 and DT_2, datasets DS_1, DS_2, and DS_3, and its data derivation trace. Adapted from (SOUZA, 2019).

2.2 Deep Learning Workflow

In this section, we go into the details of a DL workflow, breaking down the steps that define its structure. We explore the process of model selection, shedding light on the methodologies employed to navigate DL models. Additionally, we highlight the challenges in model selection.

2.2.1 An Overview of the Deep Learning Training Workflow

Based on DL workflows depicted in several works (AGRAWAL *et al.*, 2019; CHAI *et al.*, 2023; GHARIBI *et al.*, 2021; MIAO *et al.*, 2017a; POLYZOTIS *et al.*, 2018; SCHLEGEL

and SATTLER, 2023a; XIN *et al.*, 2018), we have assembled an adaptation that considers the different steps that compose this workflow. DL workflows are data-centric and model-centric, producing a DL model based on input raw data through a data transformation flow (SCHLEGEL and SATTLER, 2023a). Figure 2.2 presents the DL workflow through a dataflow perspective, composed of data transformations represented as rounded rectangles and datasets represented by cylinders. The workflow begins with data preparation, where preprocessing techniques refine the input data before splitting it into training and testing sets. These sets then undergo further transformations to build and evaluate DL models. The DL workflow often contains feedback loops among the different data transformations, which are characteristics of an experimentation process. Assuming that the trained DL models are being applied in a real production scenario (*e.g.* computer-aided disease detection or computer-aided diagnosis in a hospital), once the datasets of interest are defined, the data scientist has to steer the process of model selection to evaluate and choose the trained model that is going to be deployed (BOEHM *et al.*, 2019).

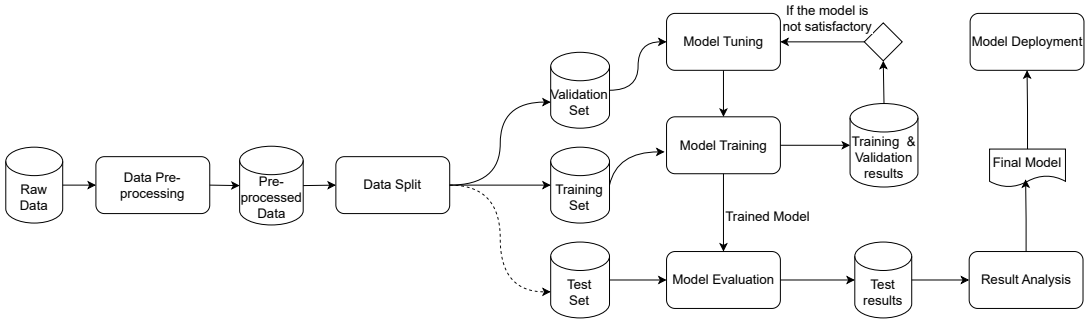


Figure 2.2: Data transformations and datasets involved in the DL workflow. Adapted from (PINA *et al.*, 2023).

The first step in the DL workflow, not depicted in Figure 2.2, is the problem definition process. During this phase, the specific problem to be addressed using DL is formulated. This entails identifying the available data, the nature of the task to be executed, and the intended outcomes. SCHLEGEL and SATTLER (2023a) characterize this step as the *Requirements Stage*, wherein the essential criteria for the model's development are deduced in alignment with the application's demands and goals.

Once the problem is defined, the subsequent step is to prepare the data that will be used to train the DL model. SCHLEGEL and SATTLER (2023a) define this step as the *Data-oriented Stage*. It involves collecting data, either from different sources or by using existing datasets (not shown in Figure 2.2 for simplicity), and preprocessing the data collected from the different sources by applying several operations, such as handling missing values, normalizing data, or removing noisy and inaccurate records. For image datasets specifically, preprocessing might involve resizing, normalization, data augmentation (*e.g.*, rotation, flipping, or cropping), and converting images to a format suitable for DL model

training. The quality of data preparation significantly influences the model's performance, prompting data scientists to employ various combinations of operations (*i.e.* preprocessing pipelines). Additionally, feature engineering and selection are performed to extract and select features for ML models. However, for DL, this step is intertwined with model training, highlighting one of the key advantages of DL over traditional methods.

After data preparation, the prepared data are split into training, validation, and testing sets. Although Figure 2.2 shows the data split occurring after preparation, in some settings, it may be necessary to divide the data into training and testing sets prior to preprocessing, depending on the specific requirements. This alternative path is represented in Figure 2.2 by a dotted arrow from the data split transformation to the test set, indicating cases where the data is split first, and preprocessing is applied to the training set, without involving the test data. Model design, training, tuning, and evaluation are contemplated in this step, which is the *Model-oriented Stage* (SCHLEGEL and SATTLER, 2023a). Designing a model involves selecting an appropriate architecture for the problem, implementing it using an ML framework, and training it on the collected and preprocessed dataset. Once trained, the model is evaluated on the test dataset to assess its performance. It is noteworthy that in critical domains, such as medical diagnosis, this step can also include extensive human evaluation. This evaluation helps assess the model's ability to generalize to new data. By analyzing training and validation results (*i.e.* DL Workflow Results), the data scientist may decide to accept the generated model, or if the model's performance is not satisfactory, it needs to be optimized by tuning the DL workflow configuration. This process is repeated until the model achieves the desired level of performance.

The next step, as outlined by (SCHLEGEL and SATTLER, 2023a), is the *Operations Stage*. A number of models are generated after the rounds. Consequently, within this stage, the data scientist faces the critical task of model selection, identifying the most suitable candidate for deployment in real-world applications. Once the chosen model is deployed, it becomes imperative to continuously monitor its performance and conduct any updates or maintenance to ensure its performance over time.

2.2.2 Model Selection in the Deep Learning Workflow

The model selection step plays a key role in the DL workflow since it allows the data scientist to obtain a prediction function that, based on a given dataset, presents "satisfactory" values for a given metric defined by the data scientist, *e.g.*, accuracy. The values of metrics, such as the accuracy of a trained model, heavily depend on the chosen DL architecture and hyperparameter values during the training step (KUMAR *et al.*, 2021). This process typically involves substantial exploratory analysis combining data preprocessing steps with training numerous architectures, as well as tuning hyperparameters to improve

the performance of the generated model (BOEHM *et al.*, 2019). Therefore, the model selection step in the DL workflow involves analyzing data from several DL workflow steps (*i.e.* data transformations). *"Model selection is not simply about reducing error; rather, it is about producing a model that has a high degree of interpretability, generalization, and predictive capabilities"* (BRUNTON and KUTZ, 2019). Model selection, a process that is both data-intensive and computationally intensive, encompasses algorithm selection and hyperparameter tuning. It is the consolidation of decisions made during these steps that ultimately determines the choice of the prediction function. This nuanced selection process has led to substantial research dedicated to the development of interpretable models and the establishment of trust in the processes involved in model training and selection, as exemplified by the works of DROZDAL *et al.* (2020); XIN *et al.* (2021).

Algorithm selection is the process of deciding which hypothesis space to use for the application. This is the decision about the type of ML classification/regression method (*e.g.* decision trees, SVMs, DNNs) and optimization procedure (*e.g.* stochastic gradient descent (SGD) or batch gradient). In this thesis, our object of study is DL. In practice, algorithm selection is not guided only by prediction metrics but rather a complex mix of technical and non-technical factors beyond just metrics, including runtime, resource cost, availability, and ease-of-use of training tools, and data scientist-specific judgments on the "interpretability" of a prediction function (BOEHM *et al.*, 2019).

Hyperparameters are key components in DL since choosing their values properly may strongly influence the quality of the outcome of an algorithm, and these values can be tuned during the multiple loops of the DL workflow in a process known as hyperparameter tuning. According to (BENGIO, 2012), a hyperparameter is defined as "*a variable to be defined before the actual application of a given learning algorithm to the data, being that variable not directly selected by the learning algorithm itself*". Tuning hyperparameters in DL is particularly challenging since the usual trial-and-error process of experimenting on them is much more expensive in complex and highly dependent models such as deep convolutional neural networks (CNNs) (HOOS and LEYTON-BROWN, 2014).

Training DL models includes several hyperparameters. Among the most common hyperparameters is the number of epochs, which defines how often an entire dataset is passed forward and backward through the network (BENGIO, 2012). When this hyperparameter value increases, the number of times the weights in a DL model are adjusted also increases, and the tendency is that the curve goes from underfitting to optimal. One has to be careful not to pass through the optimal point and reach overfitting when the number of epochs is larger than required. The batch size is the number of examples presented to the network during the training (BENGIO, 2012). Properly defining the batch size may be a hardware requirement due to the size of the GPU's memory, but it can also lead to

better generalization (MASTERS and LUSCHI, 2018; SMITH *et al.*, 2018). The learning rate hyperparameter controls how quickly or slowly the neural network weights are adjusted (BENGIO, 2012). The optimizer hyperparameter defines the algorithm or method used to adjust the network weights, for instance, by computing adaptive learning rates for each parameter (IOFFE and SZEGEDY, 2015). The output of a loss function serves as a directive for optimizers, guiding them in the process of updating the model's parameters. Multiple optimization techniques are at the disposal of data scientists. Other examples are train-test split ratio, activation function in DL model layers (*e.g.* Sigmoid, ReLU, Tanh), cost or loss function the model will use, number of activation units in each layer, the dropout rate, and the kernel or filter size in convolutional layers.

LI *et al.* (2021) provides an analysis of the current methods for exploring the DL configuration space and conducting model selection, which depends on who is in control of the learning process (LEE and MACKE, 2020; MOSQUEIRA-REY *et al.*, 2023). LEE and MACKE (2020) categorize these methods into three levels of autonomy: data scientist-driven, autopilot, and cruise-control. Based on LI *et al.* (2021) definitions, these would be manual search, AutoML, and intermittent HIL, respectively. Figure 2.3 provides a visual representation of these model selection paradigms.

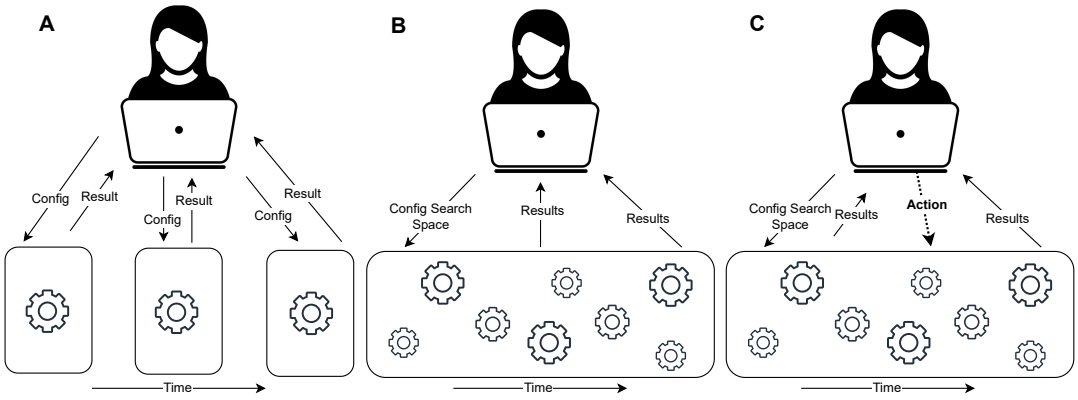


Figure 2.3: A) Manual model selection. B) AutoML-based model selection. C) Intermittent human-in-the-loop model selection. Adapted from (LI *et al.*, 2021).

Manual search model selection is an approach wherein the data scientists train a DL model by manually defining the DL workflow configuration, evaluating the DL workflow results, and correcting the model result through an interactive feedback loop, which means that the data scientist has to tune the DL workflow configuration. (HE *et al.*, 2021). In this case, the data scientist retains full control over the search process for the most satisfactory model. They will explicitly specify a configuration (or a few configurations) to explore and wait until it finishes. Based on the results of the explored configurations and human intuition about the search space, they will specify the next configuration (or set of configurations) to explore (LI *et al.*, 2021). Figure 2.3(A) illustrates this paradigm. As the data

scientist makes all the modeling decisions, including the initial workflow and iterative changes to it to evaluate in subsequent iterations (LEE and MACKE, 2020), it falls upon the data scientists to steer the entire search for the best model. This alternative is tedious and error-prone due to the lack of a systematization and consolidated record of the tested configurations. In addition, this approach presents a very low throughput (LI *et al.*, 2021), which means that very few configurations are tested.

To ease the difficulty of selecting ML models, automated machine learning (AutoML) methods have been proposed to automate several steps of the workflow, such as data preprocessing, feature engineering, model selection, and hyperparameter tuning (HE *et al.*, 2021; KARMAKER ("SANTU") *et al.*, 2021). This automation eliminates the need for human intervention, and in some instances, it is considered entirely satisfactory, rendering human involvement unnecessary. Figure 2.3(B) provides a visual representation of this approach, where the data scientist takes on a passive role, simply consuming the results generated by the explored configurations. Notable examples of such systems include Google's AutoML [1], IBM's AutoAI [2], and Auto-sklearn [3].

The last step in the DL workflow, just before deployment, is arguably the most crucial for a data scientist when adopting an AutoML approach. It involves summarizing the discoveries and suggesting the most valuable insights to domain experts. These recommendations include various aspects like models, features, or computational requirements. This step is somewhat unstructured and primarily performed manually, even when leveraging automated tools (KARMAKER ("SANTU") *et al.*, 2021). Therefore, it is important to provide data scientists with enough data and analytical resources to empower them to execute this task.

The limitations of AutoML become apparent in scenarios where human control and interpretability are crucial. This is especially problematic when a data scientist's domain expertise plays a significant role in shaping the workflow (GIL *et al.*, 2019), in high-stakes decision-making situations where trust and transparency are essential (WANG *et al.*, 2019b), and in exploratory contexts with ill-defined problems (CASHMAN *et al.*, 2018). While AutoML allows for the definition of constraints and enables users to follow its decisions, challenges remain in fully understanding the rationale behind its choices, fostering trust in the models it generates, and ensuring transparency throughout the process. On the other hand, manual search, where humans exert full control over the DL workflow, can be problematic in terms of the overwhelming number of configurations to train, making the process error-prone and complex to manage. Striking a balance between the advantages of automation and the need for human expertise and oversight remains a

[1] `https://cloud.google.com/automl/`
[2] `https://www.ibm.com/products/watson-studio/autoai`
[3] `https://automl.github.io/auto-sklearn/master/`

pertinent challenge in the field of DL. On this note, GIL *et al.* (2019) advocates AutoML systems to provide comprehensive insights, going beyond the mere presentation of the final model results.

DE BIE *et al.* (2022); KUMAR *et al.* (2021); LEE and MACKE (2020); MIDDLETON *et al.* (2022); MOSQUEIRA-REY *et al.* (2023); ROGERS and CRISAN (2023) highlight that despite AutoML's automatic support, having the human-in-the-loop (HIL) in the DL workflow is an important resource. This discussion has led LI *et al.* (2021) to propose an intermittent HIL model selection, which is a combination of the two previous paradigms, in a process that takes advantage of AutoML while allowing the human to steer the experiment in a more advanced way than simply monitoring and performing analyses with visualization tools. Figure 2.3(C) illustrates this approach. Instead of passively waiting by consuming the results of explored configurations, data scientists can steer the model selection process, creating new individual configurations or batches of configurations using a refined search space, stopping running configurations, and resuming stopped configurations.

Model selection, regardless of the chosen paradigm, be it manual, AutoML, or intermittent human-in-the-loop, presents several challenges when viewed through the lens of result analysis. Firstly, in manual model selection, data scientists face the task of curating and interpreting results, which can be time-consuming and susceptible to human biases. In contrast, AutoML aims to alleviate this burden by automating the model selection process. However, it introduces challenges related to the transparency and interpretability of the selected models, making it essential to bridge the gap between automation and understanding. Intermittent HIL attempts to strike a balance, but it introduces its own set of complexities, such as ensuring communication and collaboration between data scientists and the automated system. Consequently, in all three paradigms, the careful analysis of configurations and results remains a crucial aspect of model selection.

During the model selection process, the term *DL Workflow Configuration* refers to the set of parameters, architectural choices, and preprocessing steps defined before and during the execution of a DL workflow. This includes hyperparameters (such as learning rate, batch size, and number of epochs), model architecture specifications, and data preparation techniques. Proper configuration is important as it directly influences model performance, training efficiency, and reproducibility. The results, also called *DL Workflow Results*, encompass the outputs generated during the DL workflow, including the trained DL model, the model metrics derived from training and evaluating DL models (*e.g.*, accuracy, loss, F1-score), quantifying how well a model is performing, and the data derivation traces with any preprocessing or transformation that has been applied to make the data suitable for model training and evaluation. Additionally, these results may encompass insights into

model behavior, such as feature importance scores and fairness metrics. These results serve as the foundation for evaluating model effectiveness, guiding iterative refinements, and ensuring transparency in the decision-making process. Building on the fundamental concepts introduced earlier in this chapter, DL workflow configuration and results can be defined as follows:

Definition 2.2.1 (DL Workflow Configuration)**.** *A DL Workflow Configuration is data regarding the model configuration (e.g., hyperparameters), DL architecture and layers, and raw and preprocessed data. Each set is represented as a dataset DS in a dataflow, for example, DS_{itrain} for the hyperparameter configurations.*

Definition 2.2.2 (DL Workflow Result)**.** *A DL Workflow Result is a dataset DS_{otrain} encompassing model metrics (e.g. accuracy, loss, precision), and their data derivation traces (Definition 2.1.3).*

Human intervention adds a critical layer of expertise to this workflow. Data scientists, with their domain knowledge, actively engage in the analysis of DL workflow results. They interpret the findings, tune the DL workflow configurations, and ensure that the selected model not only excels in quantitative metrics but also aligns with the specific needs of the task at hand. To support the data scientist in these analyses, integrating the data transformations, configurations, and results within a DL workflow is crucial.

2.2.3 Existing Deep Learning Frameworks and Platforms

The development and deployment of DL models are facilitated by various frameworks and platforms that cater to different aspects of the workflow. Understanding these tools is essential for navigating the DL landscape and implementing DL projects.

DL frameworks serve as the foundational building blocks necessary for creating, training, and validating Deep Neural Networks (DNNs). They provide a range of functionalities, from defining model architectures to optimizing training processes. A key framework is TensorFlow[4], developed by Google as an open-source library for research and production use. TensorFlow supports various ML tasks and provides an ecosystem for developing DL models. Its flexibility allows developers to build complex architectures, from simple feedforward networks to advanced models like convolutional and recurrent neural networks. TensorFlow is also scalable, enabling distributed training on clusters of GPUs and TPUs, which is crucial for handling large datasets and complex models. Moreover, it offers deployment options through TensorFlow Serving and TensorFlow Lite, facilitating model deployment across cloud, mobile, and edge devices.

[4] `https://www.tensorflow.org`

Another framework is PyTorch[5], created by Facebook's AI Research lab, which has gained popularity for its user-friendly interface and dynamic computation graph. This dynamic graph allows for immediate execution of operations, making it particularly useful for tasks requiring iterative experimentation. PyTorch also provides a rich ecosystem that supports a variety of libraries, such as TorchVision for image processing and TorchText for natural language processing, allowing users to integrate specialized tools into their workflows.

Keras[6], initially a standalone high-level DNN API, is now integrated with TensorFlow. It provides a simple and intuitive interface for building DNNs, making it accessible for beginners while remaining efficient for experienced developers. Keras allows users to create complex models by stacking layers and combining different architectures, such as functional APIs for custom designs and sequential models for linear stacks. Additionally, Keras includes a library of pre-trained models, such as VGG and ResNet, which can be used for transfer learning, speeding up the training process for similar tasks.

Scikit-learn[7], while primarily a ML library, provides essential tools for data preprocessing, model evaluation, and traditional ML algorithms. It offers a wide range of algorithms for classification, regression, clustering, and dimensionality reduction, making it useful for several tasks. Scikit-learn also includes utilities for data cleaning, normalization, encoding categorical variables, and feature extraction, which are critical steps before applying DL models.

In addition to frameworks, platforms provide broader support for managing the entire ML life cycle. They facilitate experiment tracking, model versioning, and deployment, enhancing reproducibility and collaboration. Notable platforms include MLflow[8] and Weights & Biases[9], both designed to manage the end-to-end ML life cycle. They provide a suite of tools to streamline the development process, allowing users to log parameters, metrics, and artifacts during model training, thereby making it easier to compare different experiments and track performance over time. These platforms also feature a model registry that serves as a central repository for managing model versions, enabling users to store, annotate, and retrieve models easily. This feature enhances collaboration within teams by keeping track of model lineage.

DeepXDE[10] (LU *et al.*, 2021) is another platform specifically designed for scientific ML, particularly in the context of Physics-informed Neural Networks (PINNs). DeepXDE

[5] https://pytorch.org
[6] https://keras.io
[7] https://scikit-learn.org/stable/
[8] https://mlflow.org
[9] https://wandb.ai
[10] https://deepxde.readthedocs.io/en/latest/

excels in solving partial differential equations (PDEs) using DL techniques, making it valuable for simulations and modeling in scientific research. It allows users to incorporate domain-specific knowledge into their models, enabling the training of networks that respect the underlying physics of the problem at hand. Additionally, DeepXDE can integrate with popular DL frameworks such as TensorFlow and PyTorch, providing flexibility and the ability to leverage existing DL tools while focusing on scientific applications.

2.3 Physiscs-Informed Neural Networks

DNNs have been effectively applied to various problems (KIDGER and LYONS, 2020) to assist decision-making or predict the future behavior of new data. One of the recent approaches for DNNs is Physics-Informed Neural Networks (PINNs), which are revolutionizing the approaches to problems governed by partial differential equations (PDEs) in Science and Engineering (RAISSI *et al.*, 2019). Physics is informed during training by adding new components to the loss function, such as the residue of the PDE and its boundary conditions.

Proposed by Raissi *et al.* (RAISSI *et al.*, 2019), PINNs are DNNs trained to solve supervised learning tasks while respecting any given law of Physics described by general nonlinear PDEs. PINNs can effectively solve direct and inverse problems associated with PDEs, as shown in Equation 2.1.

$$\begin{aligned} \mathcal{F}(\boldsymbol{u}(\boldsymbol{z});\gamma) &= \boldsymbol{f}(\boldsymbol{z}) \quad \boldsymbol{z} \in \Omega \\ \mathcal{B}(\boldsymbol{u}(\boldsymbol{z})) &= \boldsymbol{g}(\boldsymbol{z}) \quad \boldsymbol{z} \in \partial\Omega \end{aligned} \tag{2.1}$$

The domain is defined by $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$. The vector $\boldsymbol{z}$ informs the space-time coordinates, $\boldsymbol{u}$ represents the unknown solution, and γ is the set of parameters related to Physics. The function $\boldsymbol{f}$ is responsible for identifying the problem input data, and $\mathcal{F}$ is the non-linear differential operator. The $\mathcal{B}$ operator indicates the initial or boundary conditions related to the problem, and $\boldsymbol{g}$ the boundary function. Equation 2.1 describes both direct and inverse physical problems. Direct problems aim to find the function $\boldsymbol{u}$ for all $\boldsymbol{z}$ while γ is the set of Physics-specific parameters. As for the inverse problems, γ is also determined from the data (CUOMO *et al.*, 2022).

In PINNs, $\boldsymbol{u}(\boldsymbol{z})$ is estimated using a DNN with a set of parameters $\boldsymbol{\theta}$ so that $\hat{\boldsymbol{u}}_\theta(\boldsymbol{z}) \approx \boldsymbol{u}(\boldsymbol{z})$. PINNs approximate PDE solutions by training a DNN that aims to minimize a loss function that incorporates physical knowledge of the problem, such as terms that reflect the boundary conditions, domain, and PDE residence at selected points in the domain.

Inspired by classical DNNs (MIAO *et al.*, 2017b), in Figure 2.4 we propose a specification for the PINN workflow. The PINN workflow also involves model configuration, data preparation, splitting sets into training, validation, and test sets, starting training, testing within an interval, and making steering (tuning) actions during training. Similarly, the data scientist also adjusts the model and repeats the whole process until the results are satisfactory.

Our adaptation of the classic workflow for PINNs starts with modeling the problem, which is governed by a function $f(x)$ representing a PDE, and determining the way Physics will be constrained into the DNN. We represent the process of "Inform Physics" by including the activation functions that best serve the problem, defining boundary and initial conditions, the methods to generate collocation points, loss function components, optimizer, etc. Since the problems that PINNs aim to predict do not have large amounts of data, the user might use methods to generate and regularize data through mathematical simulations. In the regularization process, the inputs and the regularization methods impact the model. In the data preparation process, PINNs may require using real data (data from the problem environment) and regularizing data using the governing PDE and/or boundary and initial conditions. The raw data format is often binary or domain-specific, so there is a data preparation process, where the data can go through a series of transformations to a format that fits and serves the model training better. This data preparation might impact the model results, so provenance helps to keep track of the whole process to allow posterior analysis and reproducibility.

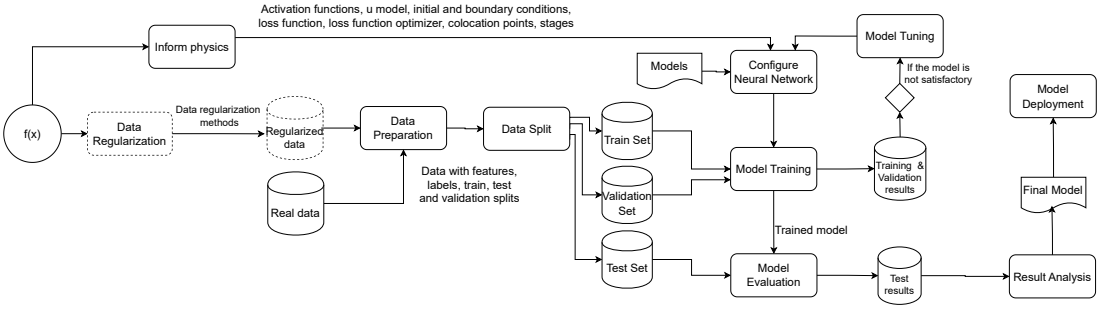


Figure 2.4: Physics-Informed Neural Networks workflow (DE OLIVEIRA *et al.*, 2023). The grey shapes are static objects that might be outputs or inputs of activities, and the white rectangles are activities (*i.e.* data transformation). The text on the edges is the possible output from an activity that may be input for the next activity. The dotted components indicate optional activities/outputs.

In a recent survey, Cuomo *et al.* (CUOMO *et al.*, 2022) present current challenges for PINN implementation, like boundary conditions management, PINN architecture design, and optimization aspects. PINNs are computing-intensive and require thousands of epochs to converge, using supercomputers, GPUs, and tools like Horovod to manage the parallel execution (STILLER *et al.*, 2020). PINNs can be designed and trained using

classic DNN libraries such as TensorFlow and PyTorch. However, tools like AutoML (KARMAKER ("SANTU") *et al.*, 2021) are incipient for PINNs, which makes PINN model selection quite complex. In addition, these DNN libraries do not automatically register and track the Physics components of the loss function. These values are essential to evaluate the accuracy of the trained model. To analyze hyperparameters with these new PINN loss components, users are required to manually define configurations to capture, register, and integrate them into the other metrics (WANG *et al.*, 2019a). For instance, to track the values of the loss components, the PINN experts have to configure tools such as TensorBoard, using the TensorFlow data summarization routines. This configuration requires programming a script to extract and aggregate information from different sources and persist it. Furthermore, in choosing the best PINN model, it is necessary to plan the organization of these files/directories of several model metrics and configurations. Aggregating these data for querying over different PINN models is far from trivial. In addition, data scientists could create their data representation (ad-hoc representation), generating heterogeneity that leads to extra work to compare trained models.

In the case of PINNs, data scientists typically steer the execution, and the support for their participation in the workflow and decision-making is even more important. One way to help track the PINN's training process is to use a provenance service, to capture the hyperparameters and PINN metrics, persist data with a DBMS, and run queries. Provenance data describes the training execution by relating data, process, parameters, hyperparameters, and metrics following a standard schema. However, most provenance systems are not prepared to manage concurrent provenance calls from distributed and parallel sources in a supercomputer or cloud environment (WOZNIAK *et al.*, 2022a).

2.4 Provenance

In scientific experimentation, scientists have to follow a series of steps, which represent the set of data transformations that scientists iterate throughout an experimental study (DEELMAN *et al.*, 2009; MATTOSO *et al.*, 2010). According to (MATTOSO *et al.*, 2010), scientific experiments include the experiment definition when scientists choose the methods to support their hypothesis (*e.g.* in DL, this step would be choosing to perform a classification experiment using DNNs with a set of preprocessing operations and hyperparameters); the experiment execution, when scientists configure the scientific workflows that correspond to their experiment definition (*e.g.* in a manual search in DL, the data scientist would write a script using TensorFlow). In this step, workflow programs (in DL, which would be the libraries and frameworks) are chosen for the methods defined in the experiment so that the workflow can be executed. Finally, the last one is the analysis phase, when the scientists can analyze the different tests for their hypothesis (*e.g.* the

accuracy obtained with different hyperparameter configurations). These different trials should be registered since it is important to consider the decisions made during the composition phase and execution results for every trial. Therefore, as the number of parameters and their variations can be large, as well as the algorithms and programs, supporting the management of the data transformations of the experiment becomes a fundamental issue.

Provenance data (FREIRE *et al.*, 2008; HERSCHEL *et al.*, 2017; MOREAU and GROTH, 2013) constitute a natural solution to assist data scientists in managing different data transformations of experiments, the data derivation trace, metadata, and parameters relevant to the data transformation steps (HUYNH *et al.*, 2019). Provenance data have already been successfully used in many scenarios and domains over the last decade (*e.g.* bioinformatics (ALMEIDA *et al.*, 2019; OCAÑA *et al.*, 2015), health (CORRIGAN *et al.*, 2019; FAIRWEATHER *et al.*, 2020), visualization (FEKETE *et al.*, 2020), *etc.*). Provenance in Computer Science was first formalized in databases (BUNEMAN *et al.*, 2001), and it has expanded for workflows (DAVIDSON and FREIRE, 2008) and scientific experiments (FREIRE *et al.*, 2008).

HERSCHEL *et al.* (2017) defines provenance as any type of information that describes the process of producing an end product, which can be anything from a piece of data to a physical object. In the early 2000s, BUNEMAN *et al.* (2001) defined provenance, also known as "lineage", as the description of the origins of a piece of data and the process by which it arrived in a database, *i.e.* provenance information describes the origins and the history of data in its workflow. The World Wide Web Consortium (W3C) defined provenance as "*information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability, or trustworthiness.*"

Focusing on methods and systems providing provenance for derived data resulting from complex data processing pipelines, HERSCHEL *et al.* (2017) observes different types of provenance that can be collected: provenance metadata, information systems provenance, workflow provenance, and data provenance. Provenance metadata encompasses any possible metadata about a production process without constraints or assumptions. Information system provenance focuses on processes within an information system responsible for disseminating data. It exploits the input, output, and parameters of a process since it lacks insights into internal processes. Workflow provenance narrows this focus to production processes represented as directed graphs (where nodes represent arbitrary functions or modules in general with some input, output, and parameters, and where edges model predefined dataflow or control flow between these modules), offering more detailed insights and requiring increased instrumentation. Data provenance, on the other

hand, tracks the processing of individual data items with the highest level of detail, necessitating structured data models and well-defined semantics for operators. The extensive instrumentation provides data provenance at the individual data item level.

The hierarchy of provenance types, depicted in Figure 2.5, follows a continuum from the most general, which is provenance metadata, to the most specific, which is data provenance. As we ascend this hierarchy, each level imposes additional constraints on the type of provenance, offering a narrower scope for collecting it. These constraints pertain to the supported processes and provenance models, limiting the available degrees of freedom for provenance collection. At the lower end of the spectrum, there may be little to no instrumentation available, resulting in ad-hoc metadata collection for arbitrary production processes. Conversely, data provenance operates in a highly instrumented environment, using the clear semantics of individual processing steps and their relationships, driven by the constraint to structured data models and associated declarative query languages. It is worth noticing that the transition from one type to another occurs gradually, without distinct boundaries.

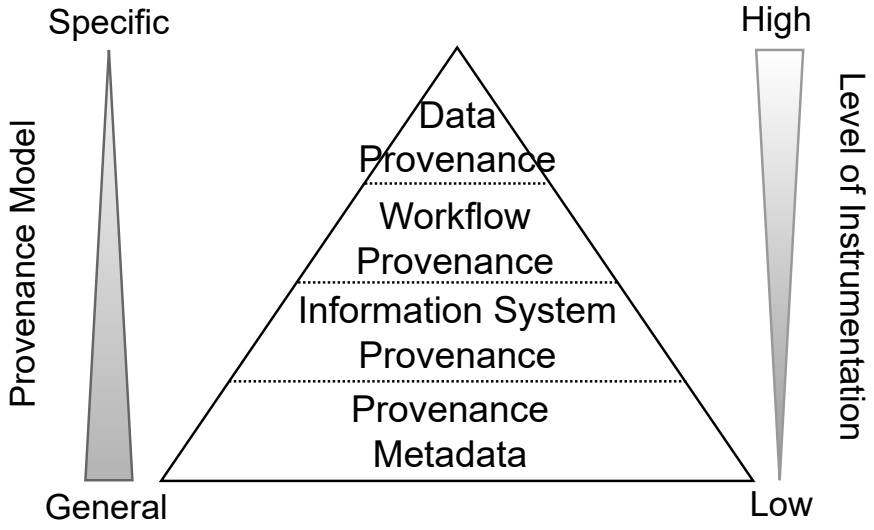


Figure 2.5: Provenance Hierarchy. Adapted from (HERSCHEL *et al.*, 2017).

CHENEY *et al.* (2009) proposed a classification for provenance based on the form used to describe relationships between data in the input and in the output: by explaining *where* output data came from in the input, showing inputs that explain *why* an output record was produced or describing in detail *how* an output record was produced.

In addition, provenance solutions generate outputs with different levels of detail, which is called granularity. There are two granularities in which provenance can be managed, known as coarse- and fine-grained (FREIRE *et al.*, 2008). Coarse-grained provenance is defined by describing the origins of a piece of data according to dataset operations,

i.e. it comprises the management of data transformations regarding the consumption and production of datasets, without considering the management of the dataflow at the logical level. This is a setup mostly adopted when considering "black box" applications in which the internal behavior is not clear. In this case, the provenance solutions cannot provide the derivation traces of separate data elements (HERSCHEL *et al.*, 2017). On the other hand, fine-grained provenance can provide the derivation traces of individual data elements in each transformation. Therefore, this granularity considers data transformations from the point of view of the flow of data elements.

Provenance data can be classified into three categories: prospective, retrospective (FREIRE *et al.*, 2008), and evolution (HERSCHEL *et al.*, 2017). Provenance is intended to describe the structure of the experiment and the abstraction of the dataflow representing the chaining of tasks through data transformations, datasets, and their dependencies. In this way, prospective provenance captures the specification corresponding to the steps that must be followed to generate data (CLIFFORD *et al.*, 2008). Regarding retrospective provenance, we assume that it is the capture of data associated with the execution of each data transformation. Thus, retrospective provenance captures the information about a workflow execution, as well as information about the environment used (CLIFFORD *et al.*, 2008). Retrospective provenance also preserves information on the resources accessed or generated during execution. It is important to emphasize that the retrospective provenance is constructed using information captured during execution, including parameter values, datasets produced, and start and end time of execution (MURTA *et al.*, 2014). Evolution provenance reflects the changes made between two versions of a dataset. Whenever a dataset is altered, the provenance solution keeps track of those changes.

2.4.1 W3C PROV Recommendation

To facilitate the exchange between provenance systems, the W3C community proposed a recommendation known as PROV[11], which has become a *de facto* standard. The W3C PROV Ontology (PROV-O) is a representation of the PROV Data Model (PROV-DM) (BELHAJJAME *et al.*, 2013), rich for representing provenance data across multiple domains and applications. It offers a high level of abstraction that can be tailored to meet the specific needs of these different domains. Figure 2.6 illustrates the core elements of PROV-DM and their relationships as arrows, following PROV specifications. PROV-DM provides an abstract representation of relationships that allows for provenance data derivations. According to PROV-DM, an *Agent* refers to accountability associating humans or platforms responsible for generating entities and for activities that happened; an *Activity* refers to a data transformation and the time at which they were created, used, or ended; and an *Entity* refers to data objects (BELHAJJAME *et al.*, 2013). The orange pen-

[11] `https://www.w3.org/TR/prov-overview/`

tagon represents the Agent concept, yellow ovals represent the Entity, and blue rectangles represent the Activity.

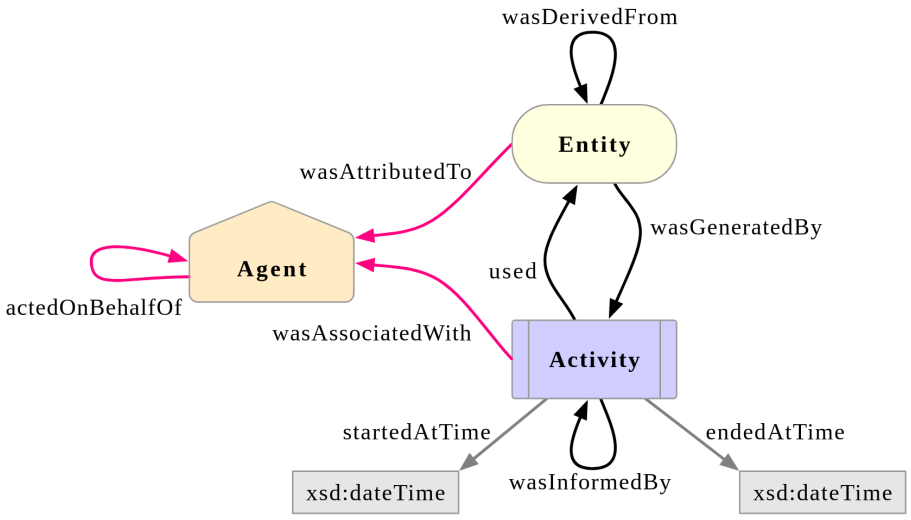


Figure 2.6: PROV-DM: The W3C PROV Data Model (BELHAJJAME *et al.*, 2013).

W3C PROV offers a flexible and rich model for representing provenance data in multiple domains and applications due to its high level of abstraction. This abstraction allows users to associate stereotypes with core entities. However, accurately modeling the relationships that describe the history of transformations is a complex task. If these relationships are not carefully considered from the start, reconstructing them later becomes challenging. Using the W3C PROV model assumes that such relationships will be established and instantiated alongside the workflow. In workflows involving sequences of data transformations, it is important to specify not only the intended flow (or prospective provenance) but also how the execution occurred in practice (retrospective provenance). The relationships in retrospective provenance are particularly difficult to capture retroactively, which is why designing with them in mind from the beginning is important. While W3C PROV gives users the freedom to model entities as needed, the provenance graph only becomes meaningful if the corresponding relationships are explicitly represented.

To present some of these concepts that will be used throughout, let us consider an example where a researcher named Alice Smith (an agent) is writing a paper (an entity) based on data from an experiment (another entity). The researcher uses specific software (another agent) for data analysis (an activity), leading to the creation of the final paper. In this example, illustrated in Figure 2.7, *PaperDraft* is an entity, *PaperWriting* is an activity, and *Researcher* is an agent.

Provenance graphs, such as the one in the example shown in Figure 2.7, can be textually represented using the PROV-N Notation (PROV-N)[12], a syntax designed to allow serializations of PROV-DM instances. The notation adopts a functional-style syntax consisting

[12] https://www.w3.org/TR/prov-n/

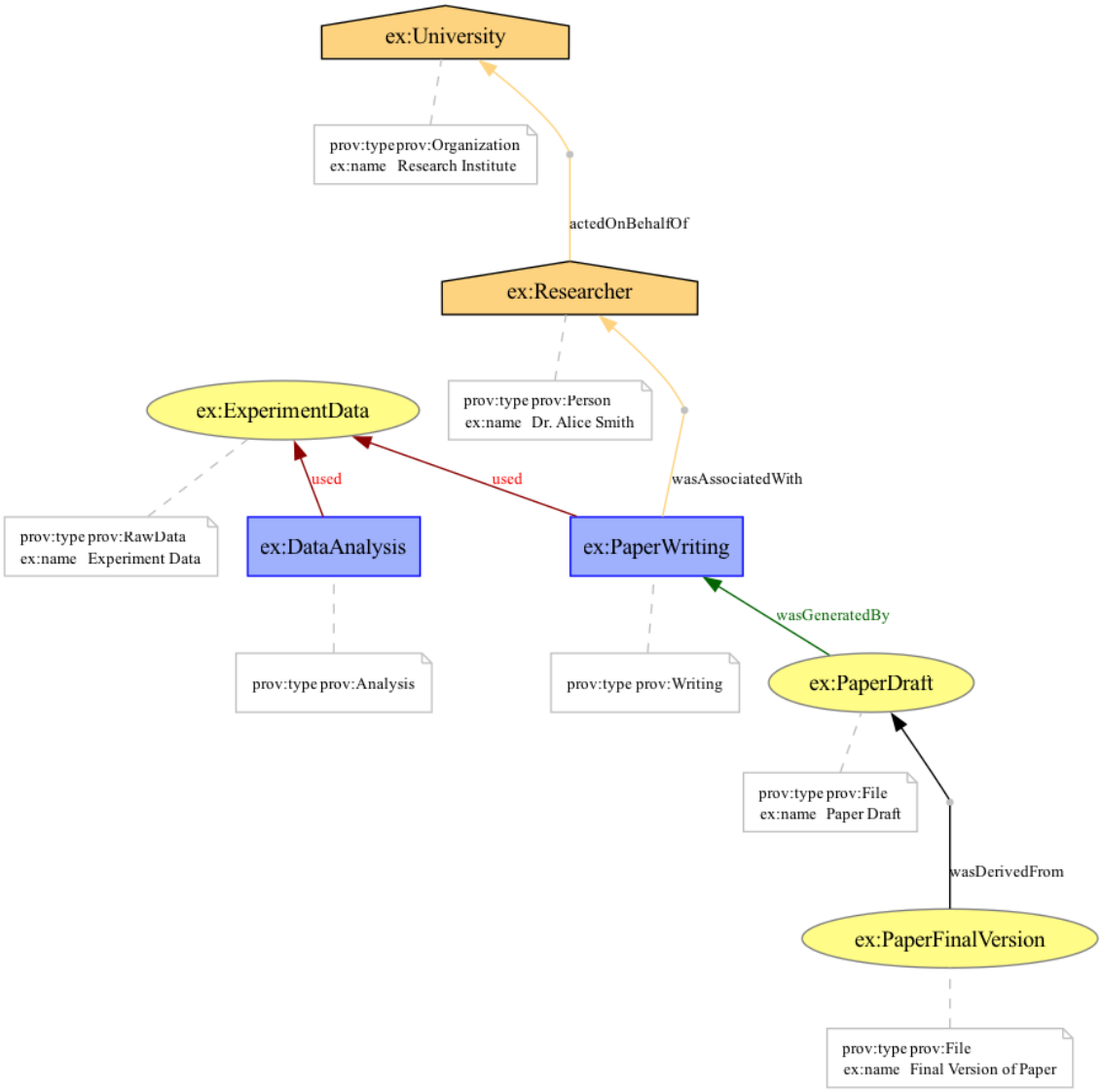


Figure 2.7: Provenance graph example following W3C PROV concepts.

of a predicate name and an ordered list of terms. Some of the expressions also allow the definition of attributes to characterize them. One important attribute, "prov:type", provides a way of categorizing entities, for example, *ExperimentData* is defined as an entity of type "RawData". The expressions in the PROV-N fragment shown in Listing 2.1 represent some parts of the provenance graph in Figure 2.7.

```
1 entity(PaperDraft)
2 entity(PaperFinalVersion)
3 entity(ExperimentData)
4 activity(PaperWriting)
5 activity(DataAnalysis)
6 agent(Researcher)
7
8 wasGeneratedBy(PaperDraft, PaperWriting)
```

```
9  wasDerivedFrom(PaperDraft, PaperFinalVersion)
10 wasAssociatedWith(PaperWriting, Researcher)
11 used(DataAnalysis, ExperimentData)
12 used(PaperWriting, ExperimentData)
13 actedOnBehalfOf(Researcher, University)
```

Listing 2.1: PROV-N fragment related to the provenance graph in Figure 2.7.

According to PROV-DM (BELHAJJAME *et al.*, 2013), there are seven core relations of provenance: generation (*wasGeneratedBy*), usage (*used*), communication (*wasInformedBy*), derivation (*wasDerivedFrom*), attribution (*wasAttributedTo*), association (*wasAssociatedWith*), and delegation (*actedOnBehalfOf*). The *used* relationship indicates "the beginning of utilizing an entity by an activity", while *wasGeneratedBy* indicates that "the completion of production of a new entity by an activity" (BELHAJJAME *et al.*, 2013). In the example from Figure 2.7, *PaperDraft wasGeneratedBy PaperWriting*. The *wasInformedBy* relation implies the exchange of unspecified entities between two activities, where an activity uses an entity generated by another activity. The *wasDerivedFrom* relation is the transformation of an entity into another entity, an update of an entity resulting in another entity, or the construction of a new entity based on pre-existing entities (BELHAJJAME *et al.*, 2013)). In our example, we see that *PaperFinalVersion wasDerivedFrom PaperDraft*.

The relations *wasAssociatedWith*, *wasAttributedTo*, and *actedOnBehalfOf* represent exchanges involving agents. The *wasAssociatedWith* relation assigns responsibility to an agent for an activity, indicating that the agent had a role in the activity. In our example, *PaperWriting wasAssociatedWith Researcher*. The *wasAttributedTo* relation indicates that a given entity was generated by some unspecified activity that, in turn, was associated with the agent. The *actedOnBehalfOf* relation assigns responsibility to an agent to perform an activity as a delegate or representative (BELHAJJAME *et al.*, 2013). In our example, *Researcher actedOnBehalfOf University*.

2.4.2 W3C PROV Adoption

Since its release in 2013, the W3C PROV model has become essential as data-intensive fields prioritize transparency, reproducibility, and trust in data handling. It has been integrated into multiple frameworks for documenting complex workflows, such as the ISA framework (Investigation/Study/Assay) in bioinformatics, which uses W3C PROV to provide provenance tracking. This promotes transparency, enabling researchers to validate and reuse datasets more effectively (GONZÁLEZ-BELTRÁN *et al.*, 2014).

Reproducing, adapting, or repeating a bioinformatics workflow in any environment requires substantial technical knowledge of the workflow execution environment, resolving analysis assumptions, and rigorous compliance with reproducibility requirements. To address these challenges, KANWAL *et al.* (2017) offers recommendations that, combined with an explicit declaration of workflow specifications, can improve the reproducibility of computational genomic analyses. The graphical representation proposed in their study has the potential to be translated using W3C PROV, allowing for testing across various platforms. This approach could facilitate the generalization of the workflow for further extension to additional bioinformatics applications.

The EarthCube Data Discovery Studio (DDStudio), a cross-domain geoscience data discovery and exploration portal, uses W3C PROV to manage provenance, detailing each metadata enhancement step with identifiers for the subject entity, the agent (such as a DDStudio administrator, metadata enhancer, or human metadata curator) responsible for the enhancement, and the operation that produces a new entity. This structured approach creates a history for each document, allowing users to trace and understand the purpose and reasoning behind every enhancement made (VALENTINE *et al.*, 2021).

In climate science research, the Pacific Northwest Climate Analysis (PNCA) Tracker addresses several challenges in providing data to be analyzed and allows the workflow components to be mapped to W3C PROV standards (YASUTAKE *et al.*, 2015). MARTINO *et al.* (2021) introduces a methodology that leverages W3C PROV to document the provenance of hydrographic datasets. By structuring each workflow activity using the PROV model, the framework supports a clear, step-by-step account of data handling, making it easier for researchers to trace and verify hydrographic data processes. According to MA *et al.* (2014), the National Climate Assessment by the U.S. Global Change Research Program (USGCRP) analyzes the impacts of climate change on the United States. Provenance information is important in this assessment, as the findings influence public interest and policy decisions. To connect Semantic Web researchers with Earth and environmental scientists, the W3C PROV ontology is integrated into the Global Change Information System (GCIS) to improve provenance tracking. This integration of the PROV-O ontology helps the interoperability of the GCIS and its datasets, facilitating connections to external sources.

CEOLIN *et al.* (2012) explores two important components of trust assessments: reputation and provenance information. They propose and evaluate a procedure for computing reputation and one for computing trust assessments based on provenance information represented with the W3C standard PROV.

2.4.3 The Role of Provenance in the Deep Learning Workflow

Human intervention adds a critical layer of expertise to the DL workflow. Data scientists, with their domain knowledge, actively engage in the analysis of DL workflow results. They interpret the findings, tune the DL workflow configurations, and ensure that the selected model not only excels in quantitative metrics but also aligns with the specific needs of the task at hand. To enable such informed decision-making, integrating data transformations, configurations, and results within a DL workflow is important.

In this context, supporting data scientists during workflow execution becomes necessary. MATTOSO *et al.* (2015) state that it is necessary for a system to support *steering* of the workflow execution, *analyzing* intermediate results and provenance data at runtime, and *taking action* during the workflow execution. Monitoring enables data scientists to leverage their domain knowledge to identify scenarios for improvement. For instance, by analyzing intermediate results, they can determine the need to adjust preprocessing steps, modify hyperparameters, or even stop execution if satisfactory results have already been achieved. The ability to steer workflow execution allows data scientists to avoid unnecessary configurations, optimizing resources and time. Thus, monitoring and analyzing intermediate results enable data scientists to interact dynamically with workflow execution.

Decision-making benefits of traceability to understand relationships within the workflow. Provenance is a natural and standard solution for traceability. Unfortunately, many metadata-based approaches do not provide traceable relationships. These approaches often lack relational structures in their data models, treating metadata as isolated attributes rather than connected entities. This limitation increases the burden on data scientists, who must manually infer relationships that the system should explicitly support.

For example, in a workflow predicting whether an individual earns more than $50K annually using the Adult Census dataset (PINA *et al.*, 2023), preprocessing often involves transformations on attributes such as “Education”. This attribute is typically one-hot encoded, where binary vectors are assigned to categories like “Bachelor”, “Master”, and “Doctorate”. In metadata-based solutions without traceable relationships, the connection between preprocessing steps and training activities may be lost, making it difficult to verify the exact transformations applied to the data. In contrast, solutions that capture these relationships can associate preprocessing steps with the training activity, preserving a detailed record of the tuples used to train the model. This traceability enhances reproducibility, enabling future experiments to replicate the exact encoding and preprocessing steps. Furthermore, in production, any discrepancies, such as new or unexpected categories in the “Education” attribute, can be identified based on the recorded provenance. If inconsistencies arise in the model’s performance during deployment, the provenance

metadata provides a reliable reference to verify whether preprocessing steps were applied consistently across training and production environments.

Promoting traceability consists of several key phases: Data Model, Capture, Store, and Query (Visualization/Analysis), as presented in Figure 2.8. The first phase, *Data Model*, involves defining a provenance data model, typically following an established standard like W3C PROV. In this phase, the relationships among agents, activities, and entities are specified, allowing for the abstract representation of provenance data. The data model relationships should define how entities are generated, used, and derived to enable traceability. In addition, accountability can be established by the relationships between agents and activities. During the *Capture* phase, provenance information is captured in real-time as activities are executed. For example, in a DL workflow, the training data (an entity) and the transformation steps (activities) are tracked, capturing how data evolves and which models (entities) are generated by specific training activities. The *wasGeneratedBy* and *used* relationships are commonly recorded during this phase, documenting the data transformations and the sequence of operations. Captured provenance data is then stored in the *Store* phase, often within a provenance-aware database or graph system. Proper storage ensures data integrity and supports data retrieval for analysis. Finally, the *Query* phase enables data scientists to analyze provenance information through querying and visualization tools. By tracing relationships and gaining insights into workflow processes, data scientists can identify bottlenecks, optimize configurations, and ensure compliance with governance and accountability standards.

When provenance is integrated directly within ML frameworks, it allows for the automatic capture of provenance data during workflow execution, reducing the overhead associated with manual intervention. However, this tight coupling can lead to challenges in flexibility, as the provenance capture capabilities may be limited by the framework's architecture and functionalities. Additionally, changes or updates to the framework may disrupt provenance tracking, potentially impacting reproducibility and trust in the captured data. On the other hand, using external provenance systems allows for greater flexibility and can facilitate the incorporation of provenance across arbitrary execution frameworks. This approach can be beneficial in heterogeneous environments where multiple ML frameworks are in use. However, it often requires additional effort to implement and maintain, as it may involve manual instrumentation to ensure that relevant provenance information is captured. This can introduce complexity and the potential for errors if not managed carefully.

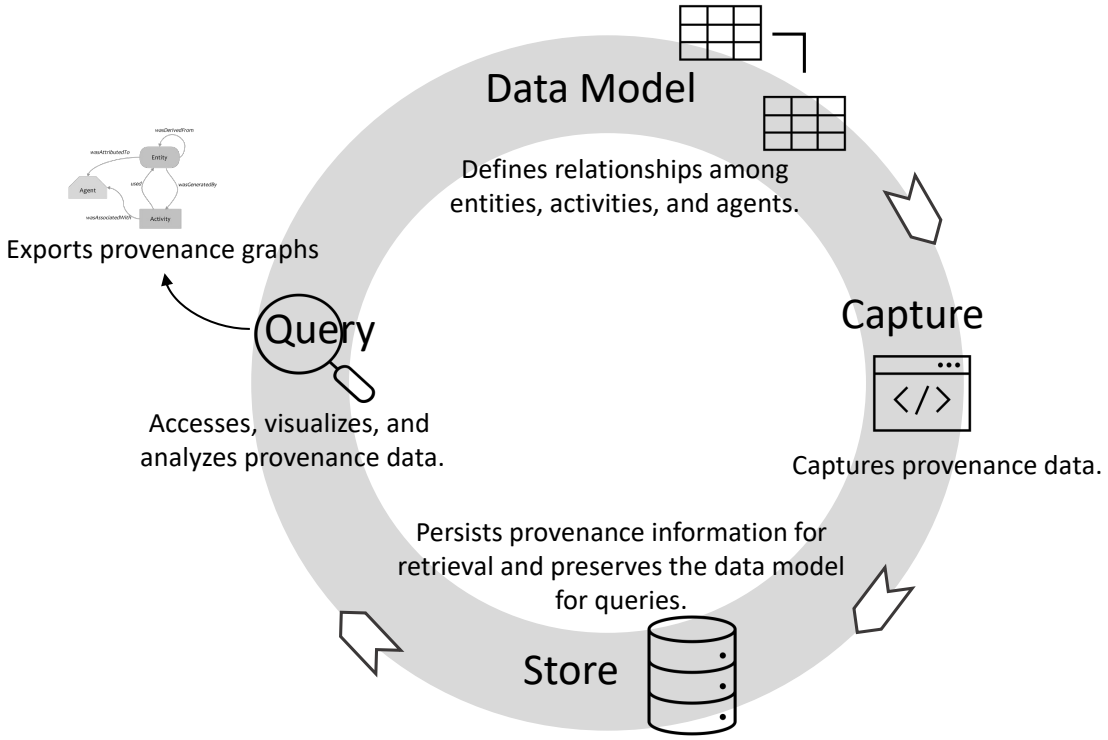


Figure 2.8: Tracking entities, activities, and relationships throughout the DL workflow.

2.4.4 Transparency, Reproducibility, and Trust in DL Workflows

Multiple audience profiles are involved in DL workflows, each with unique roles, goals, and requirements (BARREDO ARRIETA *et al.*, 2020; LANGER *et al.*, 2021). These profiles include developers and data scientists who are focused on building, training, and optimizing DL models. They require detailed insights into model performance and reliability, along with information to analyze results, identify areas for improvement, and refine the model. Domain experts and model users rely on model outputs in their work. For these users, trust in the model is critical, as they need assurance that the model's decisions are accurate and interpretable within their specific field. Another important group includes individuals affected by model decisions who require transparency and accountability, particularly in areas where fairness and ethical considerations are essential. Regulatory entities and agencies also play a key role in DL workflows, ensuring that models comply with legal and ethical standards. These bodies need access to documentation that demonstrates adherence to guidelines and allows verification that best practices in model development have been followed. Additionally, managers and executives involved in decision-making and oversight need to understand the applications of AI within their organization, including its limitations, business implications, and compliance with corporate policies.

Each of these audience profiles interacts with the DL workflow differently and brings specific priorities. For instance, model users are focused on trust, whereas developers aim to optimize model performance (BARREDO ARRIETA *et al.*, 2020). Regulatory agencies prioritize compliance with legislation, and individuals affected by model decisions require assurances that the model operates fairly and transparently. In this thesis, the focus is specifically on developers and model users.

Developers, including data scientists and ML engineers, are involved in the initial stages of model generation, including data preparation, model selection, hyperparameter tuning, and training. They benefit from transparency, as it allows them to analyze model behavior and identify potential improvements, and from reproducibility, which enables them to experiment with different configurations and settings. For developers, provenance provides a systematic approach to track every detail of the DL workflow, helping ensure that all changes and their impacts are well-documented and can be traced back. This facilitates debugging, iterative model improvement, and compliance with organizational standards and requirements for accountable ML development.

End users who apply models for predictive tasks but do not directly interact with the model's development often rely on trust as a primary concern. Examples include clinicians using predictive models in healthcare or analysts applying models in financial forecasting. For these users, provenance helps establish trust by offering transparency about how the model was developed, validated, and tested, along with information on the data sources and configurations used.

Transparency allows users to trace specific outcomes back to workflow activities, inputs, and intermediate outputs, making it possible to assess whether the model's predictions align with domain knowledge or expected behavior (HOFFMANN and EBRAHIMI POUR, 2024). It emphasizes the traceability of each decision and transformation within the workflow, providing a narrative that shows how results were achieved. Reproducibility ensures that the model's results can be consistently replicated, which is essential for developers and regulatory bodies who require evidence of model stability and reliability (DAVIDSON and FREIRE, 2008). Trust, built on transparency and accountability, is particularly critical for those affected by model decisions and organizations aiming to implement ethical AI practices (SCHERZINGER *et al.*, 2019).

Provenance data plays a central role in supporting these qualities by recording DL workflow configuration and DL workflow results (LEO *et al.*, 2024). It enhances transparency by providing a detailed trace of each decision, aids reproducibility by documenting configurations and inputs, and fosters trust by making the workflow fully transparent and auditable for all involved parties (HOFFMANN and EBRAHIMI POUR, 2024).

2.5 Exporting Provenance Graphs for Production

In DL workflows, traceability is important in debugging and improving workflows and enhancing analytical capabilities during model development, selection, and integration into production environments. Two common approaches for capturing traceability information are log files and provenance graphs. This section details the strengths and limitations of each approach, comparing their roles in DL workflow analysis and providing essential features to support trust, transparency, and reproducibility in DL models.

Log files are easy to implement and can be used to capture workflow activities and represent the workflow results as entities, but are limited in their ability to structure complex relationships (BARRINGER *et al.*, 2010). While log files can provide basic analytical capabilities and a transparent record of actions, their structure can fall short when tracking data transformations and traceability (YAO *et al.*, 2016), important for reliable reproducibility and trust in production. Provenance graphs represent the workflow structure, capturing entities (data, models, parameters), activities, and their relationships (HERSCHEL *et al.*, 2017; MISSIER *et al.*, 2013). Typically stored in a database, provenance graphs provide detailed workflow traceability, enabling fine-grained analytical capabilities during model development and selection. This traceability enhances reproducibility, interpretability, and trust in production environments (MORA-CANTALLOPS *et al.*, 2021), where model auditability and transparent decision-making are essential.

To develop applications based on DL models, scientists commonly gather data, prepare the data, train a model with the data, and evaluate this model in a cycle as represented in Figure 2.9 (SCHLEGEL and SATTLER, 2023a). Depending on the evaluation results, the model is retrained (revised by varying hyperparameters), or the data has to be revised. To move from one cycle step to another, the scientist evaluates intermediate data and logs data generated by each step. At a large scale, after many cycles, it becomes difficult to browse and analyze data from all the steps with different file formats and representations, as shown in Figure 2.9a (KUNSTMANN *et al.*, 2021). To analyze data, programs must be written to relate these different files. Recent approaches use a provenance database to integrate data from these cycle steps and improve data analyses and decisions (Figure 2.9b).

In Table 2.1, we compare log files and provenance graphs across several criteria, highlighting their strengths and limitations for DL workflow traceability, reproducibility, and production needs. The criterion *Programming Effort* considers the amount of work required to implement, set up, and manage data capture methods. It includes tasks such as defining what data to capture, writing capture logic, and ensuring the approach integrates with existing workflows. *Compilation* refers to the need for processing data after it is

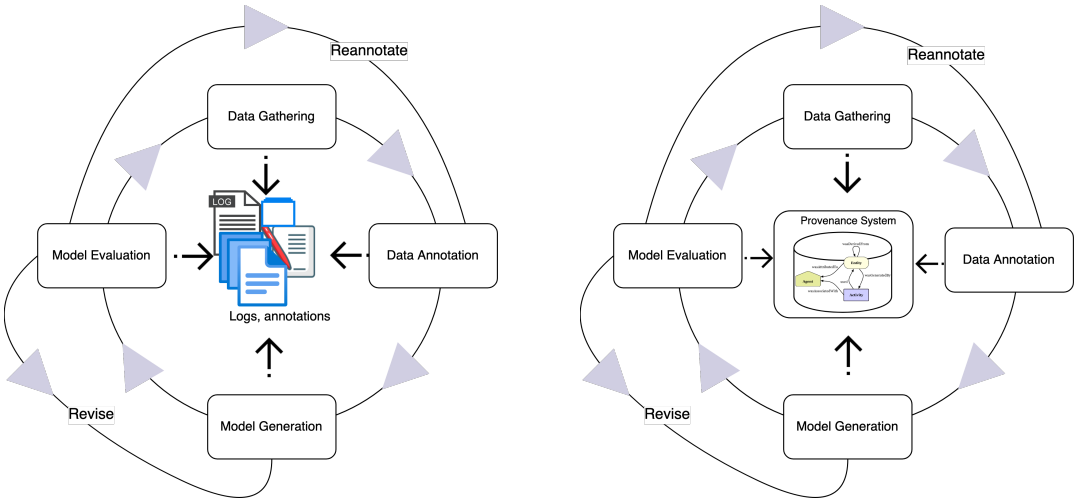


Figure 2.9: (a) Life cycle of a DL experiment. (b) Life cycle of a DL experiment using a provenance database.

captured to make it usable for analysis. Compilation may involve structuring, parsing, or transforming the data into a format that facilitates insight extraction or further analysis. The *Storage Location* criterion focuses on where the captured data is stored and how well-suited this location is for long-term storage, accessibility, and reliability in supporting complex analyses. *Data Integrity* addresses the capability to maintain accurate data over time (integrity) and to restore data to previous states (rollback), which is essential for reliable workflows and recovery after errors. *Interpretation* looks at how straightforward it is to understand the captured data, considering the clarity and standardization of the format and whether additional context or knowledge is needed for accurate interpretation. *Traceability* evaluates the approach's ability to provide a clear, chronological record of events or data states, supporting analysis by showing how data evolved throughout the workflow. The *Query Complexity* criterion mentions how easily complex questions can be answered using the data, reflecting the capability of the storage and data model to support sophisticated queries and analyses.

Table 2.1: Comparison of Log Files and Provenance Graphs.

Criterion	**Log Files**	**Provenance Graphs**
Programming Effort	Requires manual programming to capture logs and organize relevant information. Easy to implement.	Programming effort due to additional setup for generating, capturing, and managing provenance data.
Compilation	Often requires post-processing or parsing to extract information for analyses.	Generates structured data automatically, often integrated with existing systems.
Storage Location	Stored as plain files in an operating system.	Stored in an established database designed for structured data storage.
Data Integrity	Vulnerable to data loss; lacks rollback capability.	Databases provide rollback and recovery mechanisms, enhancing data integrity.
Interpretation	Dependent on user-defined log format; interpretation requires knowledge of the log format.	Standardized structure makes interpretation straightforward; relationships are inherent to the structure.
Traceability	Requires manual setup for traceability, such as sorting by timestamps to order events.	Provides inherent traceability, capturing entity relationships.
Query Complexity	Limited querying capability: complex analyses are challenging to implement.	Supports complex querying through relational or graph-based queries, simplifying complex analysis tasks.

Chapter 3

Existing Support for Traceability of Deep Learning Workflows

In this chapter, our goal is to analyze the existing literature addressing the limitations associated with providing traceability for DL workflow analyses. We have chosen to explore works related to "artifact management". For this purpose, we leveraged a survey conducted by SCHLEGEL and SATTLER (2023a), which analyzes aspects such as data analysis and lineage. The authors define artifacts to encompass elements like datasets, labels, features, data processing code, environmental dependencies, and metadata, including parameters, hyperparameters, and model metrics. Although our focus is not on artifact management, these definitions align with the data involved in our DL workflow analyses, trust, and transparency.

Managing a large number of DL models is challenging, and the research community has focused on addressing the challenges of providing resources for data scientists to manage and analyze the DL workflow (SCHLEGEL and SATTLER, 2023a). Several approaches have emerged that capture and integrate steps of the DL workflow, as evidenced by the works of (GHARIBI *et al.*, 2021; MELGAR *et al.*, 2021; ONO *et al.*, 2021; SCHELTER *et al.*, 2017; SCHLEGEL and SATTLER, 2023b; SOUZA *et al.*, 2022; VARTAK and MADDEN, 2018; ZAHARIA *et al.*, 2018). These tools are often referred to as ML (or DL) artifact management systems (SCHLEGEL and SATTLER, 2023a) and their development marks an effort to improve the management and analysis of DL workflows, providing valuable resources for data scientist.

Monitoring the training process and analyzing DL workflow steering data within an artifact management framework for a DL workflow is important to help data scientists make well-informed decisions. Observing DL workflow configurations and model metrics during the execution of model training and evaluation steps provides an understanding of

the DL model's evolving dynamics. By closely monitoring performance metrics, such as accuracy and loss, throughout the training process, data scientists gain insights into the model's learning process. This visibility allows for the identification of anomalies, overfitting, or convergence challenges, enabling data scientists to implement corrective measures or abort the training. Moreover, the analysis of intermediate data facilitates DL workflow steering actions, providing information on how different DL configurations impact the model's behavior. In essence, monitoring and analyzing intermediate data within artifact management serves as a decision-support mechanism, equipping data scientists to make informed choices throughout the DL workflow, from model refinement to the ultimate decision on model deployment.

Artifacts management for ML/DL workflow has been explored in two main directions. The first is the use of existing provenance management tools (which we call domain-agnostic), while the second leverages the development of management tools for ML/DL systems. There are several challenges in making DL workflows provenance aware, like taking into account the execution framework that may involve CPUs, GPUs, TPUs, and distributed environments such as clusters and clouds as discussed in (GIL *et al.*, 2021; SCHERZINGER *et al.*, 2019; WANG *et al.*, 2020). The following subsections delve into these two approaches.

3.1 Workflow Provenance

There are several approaches for capturing provenance data (PIMENTEL *et al.*, 2019) that can be applied (with specializations) to the DL domain. Approaches for automatic capturing provide very fine granularity, generating a significant execution overhead in the process. Systems like noWorkflow (MURTA *et al.*, 2014) capture and store provenance data from Python scripts in an automatic way. noWorkflow allows provenance data capture without requiring any modifications to the original Python script. However, it is coupled to the Python language and does not execute in distributed and parallel environments, which limits the use of parallelism in popular ML libraries. Similar to noWorkflow, SPADE (GEHANI and TARIQ, 2012) automatically collects provenance from a workflow script, including distributed and parallel environments, but this script has to be compiled using an LLVM compiler. In the automatic capture, the data scientist does not spend time defining what provenance data to capture. However, when it comes to analyzing this provenance, significant time and effort are required to understand what and how the data was modeled. In addition, due to the fine granularity, the data scientist has to filter and aggregate data before starting the analysis. Having to do this during the training cycle may not be an option.

Different from the approaches based on automatic provenance capturing, the approaches based on the participation of the data scientist allow for pre-selecting relevant data for analysis, having less impact on the execution and analysis time. When the data scientist identifies attributes and parameters in W3C PROV entities and activities with their relationships, these chosen names become familiar for runtime analysis. One of these tools is DfAnalyzer (SILVA *et al.*, 2018), which is a tool that allows data scientists to set the relevant data to be captured for runtime analysis in high-performance execution environments. Although DfAnalyzer is W3C PROV compliant and has been used in different domains, it is not designed to support provenance capture during the DL workflow. Therefore, repetitive work on designing and instrumenting has to be done. Sumatra (DAVISON, 2012) captures provenance from the script execution based on a series of annotations in the script. However, Sumatra only supports *post-mortem* analysis, *i.e.*, only after the ML model is generated. YesWorkflow (MCPHILLIPS *et al.*, 2015) is another example of a tool capable of analyzing provenance data. YesWorkflow does not depend on a programming language, adopting a strategy of adding annotations to the scripts to identify provenance data. However, its queries are based on URIs, and hyperparameter runtime analyses are not URI-based. Similar to YesWorkflow, in (GHOSHAL and PLALE, 2013), there is no runtime provenance capture. Instead, they take advantage of applications that provide log files to extract provenance from them. This could be adopted by systems like TensorFlow and Keras, which provide provenance logs. However, the queries are limited to the logged data, and it is a *post-mortem* analysis approach.

UML2PROV (SÁENZ-ADÁN *et al.*, 2018) aims at making UML (Unified Modelling Language) based applications provenance aware automatically. It is an approach that provides a mapping strategy from UML class, State Machine, and Sequence diagrams to define an automatic code generation technique that deploys artifacts for provenance generation in an application. UML2PROV assumes the existence of these diagrams or requires that they be designed or generated through reverse engineering, which limits its use in most ML environments.

Therefore, capturing provenance for analysis in DL domains using domain-agnostic approaches may be complicated due to programming language and compiler dependencies, lack of support for provenance capturing in HPC and distributed environments, and lack of support for runtime provenance analysis.

3.2 Artifact Management

To explore the current landscape of research related to this thesis and to corroborate our hypothesis that provenance is an ally in providing DL workflow artifacts analyses, we conducted a targeted search regarding the management of ML/DL artifacts. In some cases,

artifact management focuses on the context of the DL model, so a term that is also found in the literature is *model management*. Therefore, the search was conducted using the string *("machine learning" OR "deep learning") AND ("artefact management" OR "artifact management" OR "model management")* from the year 2016 across the abstracts, titles, and keywords in the ACM Digital Library[1] and IEEE Xplore[2] libraries. These platforms were selected due to their significant influence in the field and our institutional access to their resources. Initial searches on Google Scholar yielded over 8,000 results for the same search string, prompting us to focus on the more relevant and high-quality publications in ACM and IEEE. The findings are summarized in Figures 3.1 and 3.2, where the IEEE database revealed a total of 94 publications, with a notable increase in recent years, particularly in 2024, which saw 29 new entries. Similarly, the ACM database presented a total of 59 publications, with 14 published in 2024, indicating a growing interest in the subject within both academic communities.

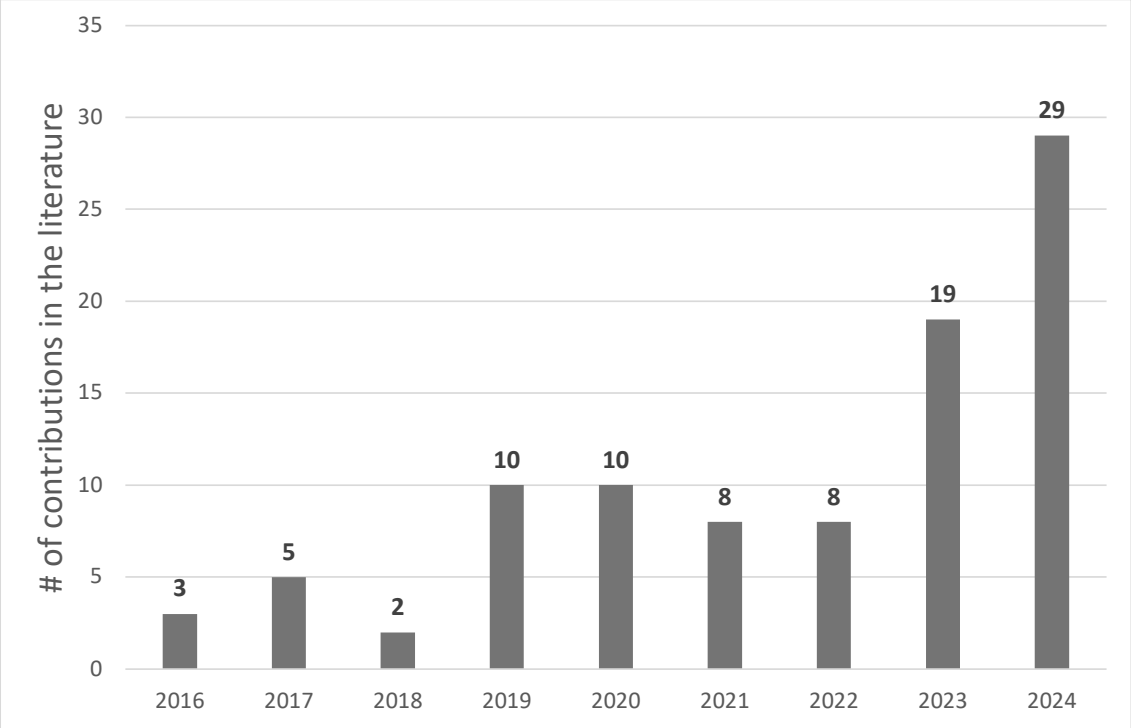


Figure 3.1: Publications found in IEEE as of 31 October 2024.

After reviewing the retrieved papers, we employed a snowballing technique to identify additional pertinent works. In addition, some of the solutions discussed in this section that did not appear in the search results were gathered from feedback provided during the review process of our submitted papers, including notable tools like Weights & Biases.

In this section, the focus is on ML systems that aim to manage the ML workflow, including training, evaluation, reproducibility, and deployment. Some of these systems orchestrate

[1] https://dl.acm.org
[2] https://ieeexplore.ieee.org/

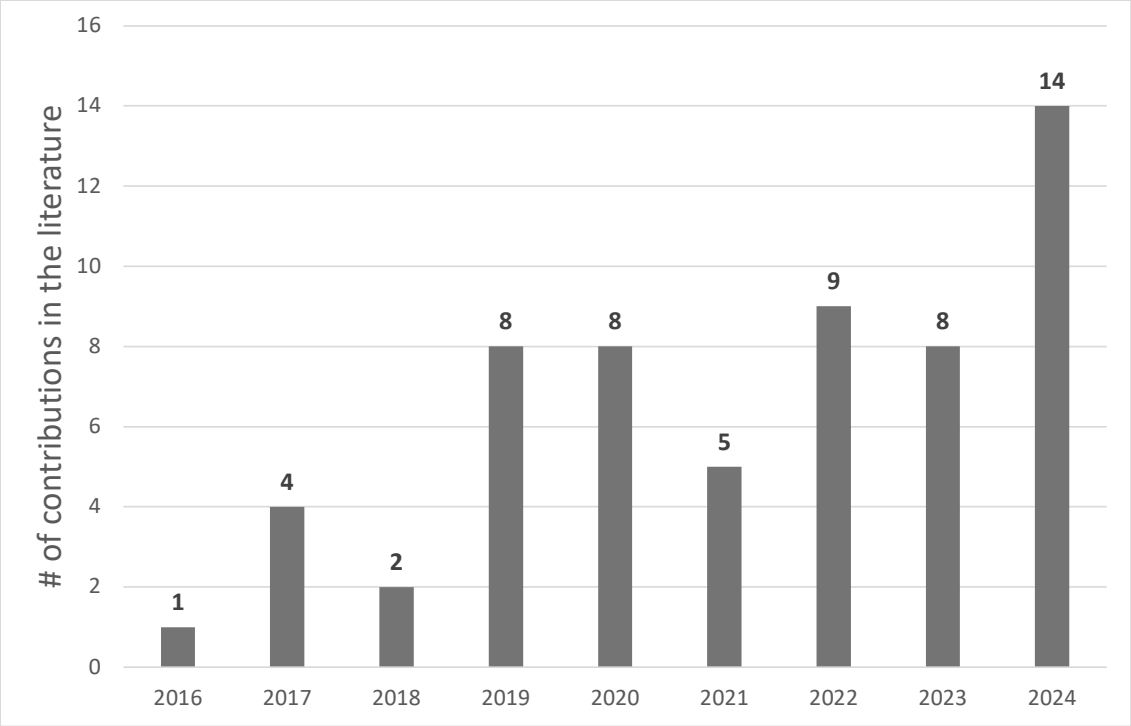


Figure 3.2: Publications found in ACM as of 31 October 2024.

the construction of ML models across all data transformations of the workflow and provide some level of provenance or artifact tracking. As these systems manage artifacts associated with the ML workflow, they can often automatically register artifacts. Other systems focus on the management of provenance or some form of tracking of artifacts within ML workflows, regardless of the solution being used to execute the ML workflow.

DL workflows generate vast amounts of data and metadata, yet many existing tools fail to capture the relationships and dependencies needed to understand and reproduce the process of training DL models. Traceability is often limited by simplistic metadata logs or systems that provide only fragmented views of workflow steps, neglecting the relationships between activities, data, and results. Without robust traceability, it becomes challenging to reconstruct the models' derivation trace, validate the results' reproducibility, or analyze the impact of specific workflow choices. This lack of provenance also undermines the ability to perform advanced queries and ensure transparency. We did not find any solution that delivers a provenance graph document to follow the DL model in production. These provenance documents add another level of trust in the deployed model. They can be analyzed to evaluate transparency, accountability, or reproducibility.

Current popular solutions like MLflow[3] (CHEN *et al.*, 2020; ZAHARIA *et al.*, 2018) and Weights & Biases (WandB)[4] provide user-friendly interfaces for managing ML work-

[3] `https://mlflow.org/`
[4] `https://wandb.ai/site`

flows. MLflow is an open-source framework that enables users to track experiments by embedding tracking commands within Python scripts. It logs parameters, metrics, and artifacts, with storage options ranging from local files to SQLAlchemy-compatible databases or remote tracking servers. Despite its widespread adoption, MLflow lacks native support for provenance traces and relationship-driven representations. As a result, it does not generate provenance documents to accompany DL models when they are deployed. WandB also offers a solution for managing ML models, from the experimentation phase to model production. However, despite allowing the definition of some relationships between artifacts, WandB requires scripting for a more in-depth analysis to traverse the resulting graph, in addition to requiring prior knowledge of the graph depth. During hands-on experience with WandB, we also identified some issues concerning its network requirements. When WandB is executed, it requires access to external IP addresses. In supercomputer environments like Santos Dumont[5], the compute nodes typically do not allow external access, making it necessary to configure the firewall to allow such connections. However, in practice, this configuration is not always feasible due to security policies or technical restrictions.

These solutions are grounded in metadata-based log systems, which provide a simplistic model for recording workflow steps. These models typically suffer from limitations such as the absence of relationships between activities, representing only attributes of individual steps without a holistic view. This lack of a relationship-driven data model restricts trace representations, impacting the ability to perform robust queries and analyses.

Solutions that rely on metadata logging tools like MLflow often face limitations in offering traceability. For instance, Gypscie (DA SILVA *et al.*, 2019; DE ALMEIDA *et al.*, 2024) is a system designed to manage the ML model life cycle, with a focus on model selection. It allows users to compose ML tasks as sequences of independent activities and uses MLflow to manage artifacts. Although Gypscie represents a step forward in integrating model management at different levels, due to MLflow's constraints in capturing provenance for traceability, Gypscie inherits the same challenges and could benefit from adopting provenance as a first-class citizen to enhance transparency and traceability.

Addressing some of these gaps, MLflow2PROV (SCHLEGEL and SATTLER, 2023b) extends MLflow by generating PROV-compliant provenance graphs based on information extracted from Git repositories and MLflow. This approach enhances traceability, but its reliance on MLflow's limited captured data results in incomplete provenance support. Additionally, its provenance graph can become overloaded with Git-related details, making it challenging to perform queries that require a clear representation of workflow activities and their relationships.

[5] `https://sdumont.lncc.br/`

The Braid Provenance Engine (Braid-DB) (PRUYNE *et al.*, 2022; WOZNIAK *et al.*, 2022b) captures provenance of data and steps in ML workflows, with a focus on producing data products that are Findable, Accessible, Interoperable, and Reusable (FAIR). It manages ML model versions, enabling traceability back to raw data, including scientific datasets. Braid-DB captures provenance data at the file and version levels rather than at the record level. Although it is not fully compliant with the W3C PROV recommendation, it covers core PROV concepts and supports analytical queries.

Comet[6] is a commercial experiment tracking platform designed to help data scientists and ML engineers monitor, compare, and reproduce their experiments. It provides logging of hyperparameters, metrics, and artifacts while offering an interactive visualization dashboard. Similar to MLflow, it allows logging metadata as key-value pairs, such as metrics, parameters, and system info, through functions like `log_parameters()` and `log_metrics()`. Comet supports multiple ML frameworks, including TensorFlow, PyTorch, and XGBoost, making it versatile for several workflows. However, similar to MLflow, it lacks native support for provenance traces and relationship-driven representations.

DPDS (CHAPMAN *et al.*, 2022, 2024; GREGORI *et al.*, 2024), which stands for Data Provenance for Data Science, is an approach designed to foster explainability, reproducibility, and trust in ML models by automatically tracking granular provenance related to raw data preprocessing operations that precede model training. DPDS provenance complies with W3C PROV and ensures that the complete data preparation step is captured in detail, providing a transparent record of how the data was handled. While it also uses a DBMS for persistence, it requires specific features of Python environments, like Pandas dataframes.

MLtrace[7] (SHANKAR and PARAMESWARAN, 2022) is a data management system that offers debugging for deployed ML workflows through assisted detection, diagnosis, and reaction to ML-related bugs and errors. MLtrace allows for the automatic logging of inputs, outputs, and metadata associated with execution, and it has an interface for data scientists to ask arbitrary post-hoc queries about their pipelines.

ModelDB (VARTAK and MADDEN, 2018; VARTAK *et al.*, 2016) is an open-source system to manage ML models, tracking model metadata through the whole ML workflow, *e.g.*, parameters of preprocessing operations, hyperparameters, *etc*. While ModelDB offers traceability, it does not follow the W3C PROV recommendation for provenance data representation. Additionally, ModelDB is tightly integrated with specific ML frameworks, which can restrict its adoption across different ML frameworks.

[6] `https://www.comet.com/`
[7] `https://mltrace.readthedocs.io`

ModelHub (MIAO *et al.*, 2016, 2017a) is a system designed for managing DL models, offering a model versioning system that allows for storing, querying, and tracking different model versions. It also features a domain-specific language, which acts as an abstraction layer for searching through the model space, along with a hosted service to store, explore, and share developed models. Since it primarily focuses on the model itself, ModelHub does not capture information about other DL workflow steps, such as data preparation, limiting its scope for end-to-end traceability.

ModelKB (GHARIBI *et al.*, 2021) is a Python library focused on managing DL models with automatic metadata extraction. ModelKB stores metadata regarding the model architecture, weights, and configurations, which allows for reproducibility, querying, visualization, and comparison of experiments. The primary goal of ModelKB is to offer model management with minimal disruption to the data scientist's workflow by using callbacks to capture metadata. However, it does not follow recommendations for provenance data representation.

ProvLake[8] (SOUZA *et al.*, 2022) provides provenance services through lightweight tracking that can be easily integrated into workflow code, such as scripts. This tracking can be applied at each step of the workflow and subsequently integrated into a data lake. By using a universal identifier, ProvLake captures relationships within the data lake and offers data modeling for all workflow steps in compliance with W3C PROV. ProvLake has been used in ML workflows and offers traceability of the ML workflow. Although it operates independently of specific frameworks, ProvLake requires consistent use throughout all steps of the workflow to ensure comprehensive tracking.

Runway (TSAY *et al.*, 2018) is a prototype tool for managing ML experiments that organizes metadata about models using a hierarchical structure of Projects, Experiments, and Runs, similar to tools like MLflow and WandB. It includes visualization features to help users explore relationships between hyperparameters and performance metrics, and it supports uploading artifacts related to model training. Although Runway is designed to be agnostic to frameworks and libraries, it only provides a Python SDK for integrating with Python scripts.

Vamsa (NAMAKI *et al.*, 2020) is a system that extracts provenance from Python ML scripts without requiring changes to the data scientists' source code. It essentially analyses scripts to determine which columns in a dataset have been used to train a certain ML model, automatically recording the relationships between data sources and models at a coarse-grained level.

[8] `https://ibm.biz/provlake`

Amazon SageMaker[9] (NIGENDA *et al.*, 2022) provides resources for model building, training, deployment, and metadata tracking. SageMaker provides traceability to some extent through its built-in capabilities for tracking model metadata, such as training configurations, metrics, and model artifacts. However, this traceability is largely limited to its internal environment, with no support for exporting detailed provenance traces to external systems. SageMaker relies on proprietary formats for logging and metadata management. In addition, SageMaker is tightly coupled with Amazon Web Services (AWS), making it less adaptable for use in diverse execution frameworks. This dependency on AWS infrastructure reduces its flexibility for integrating with external tools or workflows beyond its ecosystem.

Table 3.1 assesses some of the existing solutions based on the following criteria: *Independence*, *Traceability*, *DL Provenance Graph*, *Provenance Representation*, and *Provenance Graph in Production*. *Independence* highlights whether a solution supports arbitrary execution frameworks. *Traceability* assesses whether a solution captures the necessary information so that provenance traces can be generated, encompassing the steps of DL workflows. *DL Provenance Graph* evaluates whether a solution produces a provenance graph derived from the captured traceability. *Provenance Representation* shows whether the solution adheres to any standards for provenance representation. *Provenance Graph in Production* indicates if a solution supports exporting the generated provenance graph for use in production.

Table 3.1: Assessment of solutions according to *Independence*, *Traceability*, *DL Provenance Graph*, *Provenance Representation*, and *Provenance Graph in Production*.

System	Independence	Traceability	DL Provenance Graph	Provenance Representation	Provenance Graph in Production
Braid-DB	Yes	Yes	No	N/A	No
Comet	Yes	No	No	N/A	No
DPDS	Yes	No	No	PROV	No
MLflow	Yes	No	No	N/A	No
MLflow2PROV	Yes	No	No	PROV	Yes
Mltrace	Yes	Yes	No	N/A	No
ModelDB	No	Yes	No	PMML	No
ModelKB	Yes	No	No	N/A	No
ModelHub	Yes	No	No	NNEF/ONNX	No
ProvLake	Yes	Yes	Yes	PROV	No
Runway	Yes	No	No	N/A	No
SageMaker	No	No	No	N/A	No
Vamsa	Yes	No	No	N/A	No
WandB	Yes	Yes	No	N/A	No
DLProv	**Yes**	**Yes**	**Yes**	**PROV**	**Yes**

[9] `https://aws.amazon.com/sagemaker/`

Table 3.1 shows that most existing solutions fail to provide provenance support across the steps of DL workflows. While some systems like Braid-DB, Mltrace, and WandB offer traceability, few solutions offer a complete DL life cycle provenance graph or the ability to export the provenance graphs into production. The adoption of provenance representation standards is limited, with only a few tools, such as DPDS, MLflow2PROV, and ProvLake, leveraging established standards like PROV. Independence is provided by more solutions, with more than half of the solutions supporting arbitrary execution frameworks. For instance, MLflow and WandB are versatile and compatible with a wide range of ML frameworks such as TensorFlow, PyTorch, and scikit-learn, while Vamsa facilitates provenance capture in Python scripts. DPDS also supports Python scripts, but data should be in the form of Pandas dataframes.

Chapter 4

Proposed Approach: DLProv

In this chapter, we introduce DLProv, the suite of provenance services proposed in this thesis. We present its key aspects within the traceability flow outlined in Chapter 2, covering the provenance data model, mechanisms for capturing and storing provenance, and the analysis of DLProv provenance data through queries.

4.1 Suite of Provenance Services

Generating a provenance graph is a natural solution for traceability. Therefore, we designed the DLProv suite[1] to integrate provenance services into DL workflows, capturing prospective and retrospective provenance, thereby supporting detailed analyses and enhancing trust in DL models. This suite includes the core DLProv services alongside tailored instances for Keras and PINNs, as well as support for interoperability with other tools capturing provenance at different workflow steps, including data preparation.

The DLProv suite architecture consists of four layers, according to Figure 4.1, namely: (i) *Training Layer*, (ii) *Data Layer*, (iii) *Provenance Integrator Layer*, and (iv) *Analysis Layer*. The *Training Layer* is where the training library (*e.g.* AutoML libraries, TensorFlow, Keras, PyTorch) executes and interacts with the *Provenance Extractor*, which accesses the hyperparameter values and metrics at runtime and gets their values. The *Provenance Extractor* is implemented and deployed independently of the training library. Consequently, data scientists can choose any training library or DL framework and instrument their code by adding calls to the *Provenance Extractor* component. This enables them to specify what data to capture (prospective provenance) and where in the script it should be captured (retrospective provenance). We assume that the programs in the DL workflow are gray boxes, *i.e.* part of their source code can be adapted, while the other

[1] `https://github.com/dbpina/dlprov`

part can invoke private source code (black box). By capturing both prospective and retrospective provenance, DLProv not only supports queries but also enables auditing of the captured provenance, as it allows comparing expected derivations with actual executions.

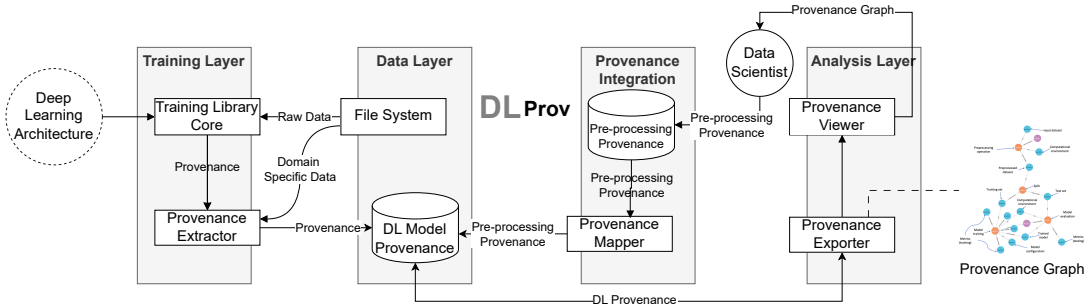


Figure 4.1: DLProv Suite Architecture (PINA *et al.*, 2025).

4.1.1 Provenance Data Model

A provenance graph is a structured representation that models the dataflow, processes, and relationships within a system. It provides an account of how data is generated, transformed, and used by capturing interactions between entities (data or artifacts), activities (processes or data transformations), and agents (people or systems). In this directed graph, nodes represent the core elements, entities, activities, and agents, while directed edges represent relationships such as *used*, *wasGeneratedBy*, and *wasDerivedFrom* (BELHAJJAME *et al.*, 2013; HUYNH *et al.*, 2018).

Consider a scenario where a DL model is being trained. The resulting provenance graph for this process would have agents, entities, and activities as nodes. Entities would be the raw dataset and any intermediate datasets produced during preprocessing, hyperparameters, and model metrics. Activities would be data preparation steps, DL model training, and validation processes. Agents would be those responsible for initiating the training process or the computational environments where the DL model training occurs. The edges in the provenance graph represent the relationships between the activities and entities. *Used* edges would connect activities to the entities they use. For instance, an edge may link a training activity to the preprocessed dataset it uses. *WasGeneratedBy* edges would indicate the generation of new entities by specific activities. A model training activity, for example, would be linked to the trained model it produces.

The data preparation can also be represented in the provenance graph, as in the example shown in Figure 4.2. For instance, consider a dataset of flower images where each image contains elements like surrounding grass or background. A cropping operation could be applied to isolate only the flower in the image. Suppose the original raw image is 1024x768 pixels, and the flower is centered within a 500x500-pixel square. The cropping

operation would extract this 500x500 region and discard the rest, creating a processed image that highlights the flower and minimizes noise in the dataset. In W3C PROV terms, the raw image is represented as an entity, named *ex:rawImage*, with metadata specifying its dimensions (1024x768 pixels) and source (Dataset A). This image is connected to a *ex:Cropping* activity through the *used* relationship, indicating that it served as input for the operation. The activity can include metadata about its start and end times, as well as specific parameters used to define the rectangular area to be cropped from the raw image. The resulting cropped image is another entity, named *ex:croppedImage*, with updated dimensions (500x500 pixels). This entity is linked to the cropping activity through the *wasGeneratedBy* relationship, indicating that the cropping activity produced this new image. Additionally, the cropped image is linked back to the original raw image through the *wasDerivedFrom* relationship, establishing that the cropped image originates from the raw image. To ensure complete traceability, the cropped image can also be connected to the aforementioned training activity, illustrating the data's flow from preparation to DL model training. The provenance graph also includes an agent, representing the data scientist who performed the operation, identified as *ex:Alice*; This agent is connected to the cropping activity with the *wasAssociatedWith* relationships, indicating responsibility for executing the operation.

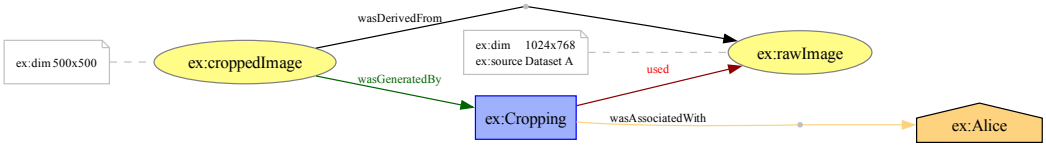


Figure 4.2: A W3C PROV representation of an image cropping activity transforming a raw image entity into a cropped image entity.

To represent the provenance information captured in DL workflows, we propose a specialized provenance data model derived from the domain-agnostic PROV-DM (BELHAJJAME *et al.*, 2013). Given that most ML workflows follow a common structure, typically involving data preparation, data splitting, model training, and testing, we adapt PROV-DM to explicitly incorporate these elements. This specialization ensures that the provenance data model captures details of DL workflows while maintaining compatibility with PROV-DM. Figure 4.3 presents the UML class diagram of the proposed provenance data model. The initial version of this model was introduced in (PINA *et al.*, 2021) and later refined in (PINA *et al.*, 2023), adding classes that highlight the relationships between data transformations within the DL workflow. This provenance data model is tailored to capture training-specific data from DL experiments while integrating them into the preprocessing transformations performed on the data.

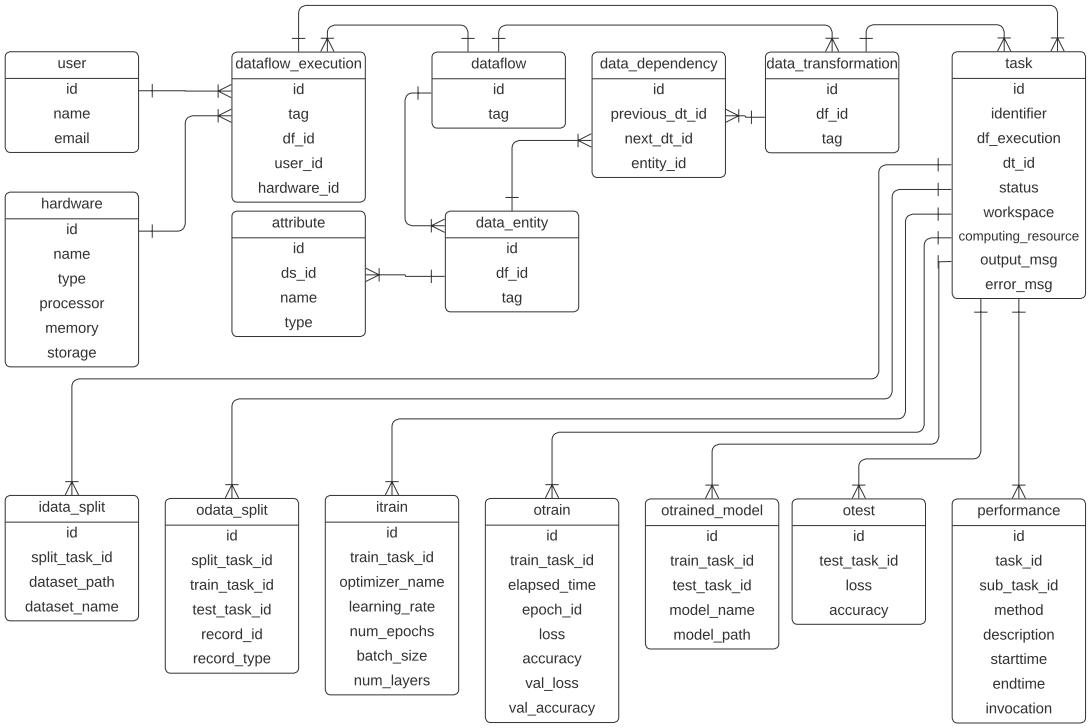


Figure 4.3: Provenance data representation for DL workflows. Extended from (PINA *et al.*, 2023).

The classes related to the dataflow specification are inherited from the PROV-Df model (SILVA *et al.*, 2016), which are *dataflow*, *data_transformation*, *data_entity*, *data_dependency*, and *attribute*. These classes represent prospective provenance. At its core, the *dataflow* class represents the workflow, identified by an id and a descriptive tag. The execution of a specific instance of dataflow is managed by the *dataflow_execution* class. Within a dataflow, individual steps, *i.e.* activities, are defined as *data_transformation*, such as *train*.

DL workflow configuration, which consists of different entities that are used or generated throughout the DL workflow, is captured by the *data_entity* class. In previous papers, we have used this class with the name *data_set* as any collection of data elements to be consumed or produced during an activity. However, to avoid confusion with a dataset that serves as input for training a DL model, we employ the term *data_entity* in the current provenance data model. The *performance* class monitors task performance, recording attributes such as method descriptions, start and end times, and invocation details. Relationships between transformations are tracked using the *data_dependency* class, which links transformations through their *previous_dt_id* and *next_dt_id*, along with the associated data entities.

To represent retrospective provenance, the provenance model contains the class *task*, which represents the instances of activities that are executed, which allows for recursive queries that trace back the execution. The input of the training process is captured in the *itrain* class, which includes parameters such as the optimizer name, learning rate, number of epochs, batch size, and number of layers. The outcomes of the training are represented by the *otrain* class, which stores metrics like loss, accuracy, validation loss, and validation accuracy for each epoch. The resulting trained models are managed by the *otrained_model* class, which includes identifiers for the associated training task, model name, file path, and related test task. Testing processes are tracked in the *otest* class, which records test accuracy and loss. Data splits used in training and testing are handled by the *idata_split* and *odata_split* classes, with attributes to identify the dataset paths, names, and task identifiers for training and testing. A DL model training or even a data preparation step can be performed on various hardware platforms, ranging from laptops to supercomputers with different GPUs. To capture this variability, the provenance model includes a *hardware* class that persists information about the hardware resources used during execution. Additionally, the provenance model supports the tracking of user information through the *user* class, which can be associated with a specific execution.

With these classes, the data scientist can discover which preprocessing methods were applied to the data used to train/evaluate a model. The prefix *i* refers to the input parameters of a task, and the prefix *o* in the classes refers to the output parameter values. This model is designed to be extensible, allowing the incorporation of additional elements, such as hyperparameters, metrics, and data transformations, as needed for more detailed analysis. For example, the activity *ex:Cropping* and the resulting entity *ex:croppedImage*, shown in Figure 4.2, could be integrated into the provenance data model to be captured.

Having a well-defined provenance model is important because it provides the foundational structure for generating a provenance graph. By explicitly capturing details about the entities, activities, and agents involved in the execution, this provenance model ensures that relevant aspects of the workflow are recorded and represented. Later, the provenance graph will be generated based on this model, allowing for detailed analysis, tracking of relationships, and accountability throughout the DL workflow. This structured representation not only enhances transparency and trust but also facilitates the interpretation and verification of results, which is important for trust in the model's outcomes.

4.1.2 Provenance Capture

DLProv operates independently of the DL framework, as its services are invoked through script instrumentation, in an extended version of DfAnalyzer (SILVA *et al.*, 2020). This approach enables the capture of the DL workflow configuration, as shown in the prove-

nance model in Figure 4.3, while also tracking dependencies within the workflow. The data scientist defines which domain data, model configurations, and model metrics will be captured and where they should be captured. With this instrumentation, the *Provenance Extractor* can access the data during training. For instance, Listing 1 illustrates a fragment of the instrumentation for a simple workflow. In this example, the workflow takes a preprocessed dataset, splits it into training and testing subsets, and then uses the training set to train a DL model. A dependency, such as *dependency=t1*, means that activity *t2* (*e.g.*, model training) can only start after activity *t1* (e.g., data splitting) has produced the required output. For simplicity, the inputs and outputs of activity *t1* are omitted. In Listing 1, lines 2 and 3 define the prospective provenance, whereas lines 5 to 18 handle the capture of retrospective provenance.

```
1  dataflow_tag = "example"
2  df = Dataflow(dataflow_tag, predefined=True)
3  df.save()
4  exec_tag = dataflow_tag + "-" + str(datetime.now())
5  t1 = Task(1, dataflow_tag, exec_tag, "SplitData", dependency = t0)
6  class DLProvCallback(Callback):
7      def on_epoch_end(self, epoch, logs=None):
8          tf2_output = DataSet("oTrain", [
9              Element([timestamp, elapsed_time, loss, accuracy,
10             val_loss, val_accuracy, epoch])])
11         t2.add_dataset(tf2_output)
12         t2.save()
13 t2 = Task(2, dataflow_tag, exec_tag, "Train", dependency = t1)
14 callbacks = DLProvCallback(t2)
15 tf2_input = DataSet("iTrain", [Element([optimizer_name,
16     learning_rate, epochs, batch_size, num_layers])])
17 t2.add_dataset(tf2_input)
18 t2.begin()
19 model.compile(optimizer=opt,
20                loss='sparse_categorical_crossentropy',
21                metrics=['accuracy'])
22 model.fit(x_train, y_train, epochs=epochs, validation_split=0.2,
23     callbacks=callbacks)
24 tf2_output_model = DataSet("oTrainedModel", [Element([trained_model,
25     trained_model_path])])
26 t2.add_dataset(tf2_output_model)
27 t2.end()
```

Listing 1: Fragment of DLProv instrumentation showing activity dependencies in a simple DL workflow.

Consider the cropping scenario. The workflow starts with a raw image entity that undergoes a cropping operation. Once cropping is applied to all raw images in the dataset,

the resulting cropped images are used in subsequent steps, such as splitting the data into training and testing sets, followed by training a DL architecture. Here, the splitting activity is identified as *t1*, and the training activity as *t2*, with *t2* explicitly dependent on the completion of *t1*. This dependency ensures that the training process cannot begin until the split images are available.

Intermediate steps, such as additional preprocessing operations applied to the cropped images before training (*e.g.*, data augmentation or normalization), would also depend on the output of the cropping operation. This creates a chain of dependencies, where each activity is linked to the successful completion of the preceding step, ensuring the workflow's traceability.

4.1.3 Provenance Store

Once captured, the provenance data is sent asynchronously with the DL execution to the *Data Layer*. DLProv adopts a DBMS as its primary persistence storage, leveraging decades of solid technology. The DLProv suite uses a DBMS to persist provenance data represented with the provenance model, according to Figure 4.3. This DBMS is currently instantiated with MonetDB[2], a columnar database optimized for analytical workloads, as its provenance storage solution.

During the training of a DL model, provenance data is persisted, ensuring minimal interference with the training process. This allows scientists to execute SQL queries on the data while the DL model is still being trained. Even after the training phase finishes, the provenance data remains stored in the database, enabling continued analysis and querying to support trust and the assessment of model performance.

In some cases, provenance capture for different stages of a DL workflow may be handled by separate solutions, for example, one solution capturing provenance during data preparation and another during DL model training. To address this scenario, and assuming both solutions adhere to the W3C PROV recommendation, the *Provenance Integrator Layer* ensures integration of the provenance information collected by these solutions. To allow the integration between these data models of two provenance databases, the *Provenance Mapper* obtains the identifier for each record or image and inserts this identifier in the *DL Model Provenance* database, along with the type of data transformation that will use the record.

[2] `https://www.monetdb.org/`

4.1.4 Provenance Queries

During the development and selection of DL models, scientists rely on queries to analyze metadata captured during training. This metadata includes details about training epochs, parameters, and the performance of candidate models, such as in Figure 4.3. By querying this database, scientists can make informed decisions about selecting the "best" candidate for deployment. Once a DL model transitions into production, queries play an important role in ensuring reproducibility and verifying consistency. For instance, scientists may use queries to confirm whether the preprocessing steps applied during model development were replicated in the production environment. These checks are important for detecting discrepancies that could compromise the reliability of the deployed model.

The *Analysis Layer* provides data scientists with analytical capabilities, enabling them to perform queries during training and explore provenance data in greater depth after training is complete. One of the advantages of provenance analyses using DBMSs is that they can be independently invoked, using interactive tools beyond traditional query languages. Most DBMSs are compatible with user-friendly dashboards and visualization platforms, such as Elasticsearch[3] and Kibana[4], which support rich, intuitive visual analyses, as shown in (KUNSTMANN *et al.*, 2021). Moreover, approaches such as text-to-SQL (NASCIMENTO and CASANOVA, 2024) can automatically generate SQL queries from natural language, lowering the technical barrier for users.

While many analyses are "well-suited" with "columnar queries", traversal provenance queries can benefit from a graph database. To meet this need, the DLProv suite includes functionality for generating provenance graphs and ingesting them into Neo4j[5], to be queried using Graph-DBMS interfaces. By being W3C PROV compliant, DLProv can use tools like PROV Database Connector[6] and PROV2Neo[7], among others. With the *Provenance Exporter*, scientists can create a provenance graph that encompasses the entire workflow, including all candidate models, or focus specifically on a single execution. To generate a provenance graph for all executions, the data scientist can use the command `python generate_prov.py -df_tag <insert df_tag>`. To generate the graph for a specific execution, the scientist can run `python generate_prov.py -df_exec <insert df_exec>`. The `<df_tag>` parameter is defined by the user in the code, and the `<df_exec>` is automatically defined by DLProv. Both values can be queried in MonetDB.

[3] https://www.elastic.co/elasticsearch
[4] https://www.elastic.co/kibana
[5] https://neo4j.com
[6] https://github.com/DLR-SC/prov-db-connector/
[7] https://github.com/DLR-SC/prov2neo

The generation of provenance graphs assumes that the data is already stored in MonetDB following DLProv's provenance model. Queries are then executed on MonetDB to retrieve entities, such as datasets, models, and hyperparameters, along with activities, which represent data transformations, and agents, including the user responsible for the DL execution and the machine used. With this information, the nodes of the provenance graph are established. Next, DLProv extracts relationships based on the provenance model, leveraging attributes like *previous_dt_id* and *next_dt_id* to infer *used* and *wasGeneratedBy* relationships between entities and activities, as well as associations between agents and activities. The provenance document is then ready to be saved in different formats, such as JSON, PROV-N, PDF, and PNG.

Once the provenance document is constructed, it can be visualized using these formats, but it can also be ingested into graph databases for further analysis through queries. DLProv provides functionality for ingestion into Neo4j, a graph database optimized for handling complex relationships. This ingestion process involves translating the W3C provenance document into a property graph model compatible with Neo4j's structure. Each node in the provenance graph is mapped to a corresponding node in Neo4j, with labels and properties that retain key metadata. Similarly, edges representing relationships between nodes are preserved to maintain the semantic integrity of the provenance data. This is achieved through solutions like the PROV Database Connector[8], which is used in this thesis. DLProv creates a database in Neo4j, reads the provenance document, and, using the PROV Database Connector, maps and ingests the provenance data while preserving its structure and relationships.

Table 4.1, adapted from (PINA *et al.*, 2024) outlines a set of queries used during two key stages of the workflow: *(i)* the training and selection of DL models, and *(ii)* their deployment and use in production. These queries were gathered from the literature and Data Science Stack Exchange (DSSE) webpage[9], and adapted from the Provenance Challenges[10], as there is currently no established benchmark for this task. While the selected queries may not cover all possible use cases, they are representative of common provenance analysis needs. The queries vary in complexity; some focus on single DL workflow activities, such as DL model training, while others trace the derivation path of artifacts across multiple stages. In the *Workflow Stage* column, we distinguish between *Development* and *Deployment* stages. A query classified under *Development* can be performed during the generation and selection of DL models, leveraging data from multiple executions and configurations. Queries marked as *Deployment* focus on the provenance data of the deployed DL model, providing insights into its behavior and derivation trace in

[8] https://github.com/DLR-SC/prov-db-connector/
[9] https://datascience.stackexchange.com/
[10] https://openprovenance.org/provenance-challenge/WebHome.html

the production environment. If a data scientist needs to analyze additional executions or configurations beyond the deployed model, the workflow is assumed to revert to the *Development* stage for further exploration. This distinction clarifies the scope and applicability of each query within the DL workflow.

Table 4.1: Examples of typical provenance queries in DL workflows.

Id	Query	Workflow Stage	Source
Q1	Find all trained models with a specific value for the learning rate.	Development	Provenance Challenge 1[1]
Q2	What was the epoch's average processing time of the model training?	Development, Deployment	(SOUZA *et al.*, 2022)
Q3	What is the hyperparameter configuration used to train the model with the highest average training accuracy?	Development	Provenance Challenge 1
Q4	What is the hyperparameter configuration used to train the model with the highest test accuracy?	Development	Provenance Challenge 1
Q5	For a given experiment, which data contributed to the run with the highest test accuracy?	Development	Provenance Challenge 1
Q6	Given a training set, what are the values for hyperparameters and evaluation measure associated with the trained model with the least loss?	Development, Deployment	(SOUZA *et al.*, 2022)
Q7	Which hyperparameters were used in this model?	Development, Deployment	(SCHELTER *et al.*, 2017)
Q8	What was the computational environment used to train a given model?	Development, Deployment	Provenance Challenge 4
Q9	Who was responsible for training a given model?	Development, Deployment	Provenance Challenge 4
Q10	Find the process that led to a given model (*i.e.* model M)/everything that caused model M to produce these results.	Development, Deployment	Provenance Challenge 1
Q11	Which data was used to train this model?	Development, Deployment	(NAMAKI *et al.*, 2020)
Q12	What feature transformations have been applied to the data?	Development, Deployment	(SCHELTER *et al.*, 2015)
Q13	High loss even tuning hyperparameters and applying data augmentation	Development	DSSE[2]
Q14	Training with imbalanced dataset not giving good validation accuracy	Development	DSSE[3]

[1] `https://openprovenance.org/provenance-challenge/FirstProvenanceChallenge.html`
[2] `https://datascience.stackexchange.com/questions/115857`
[3] `https://datascience.stackexchange.com/questions/115268`

When training a DL model with several configurations, queries can provide insights into the process and outcomes of the experiments. These queries enable comparison and selection of the most suitable model for deployment by analyzing training configurations, performance metrics, and other information. They support the evaluation of multiple training runs, helping to identify configurations that yield the best results or optimize specific criteria. For instance, Q1 helps filter the experiments. By determining the average time per epoch during training, Q2 can provide insights into the computational performance of different configurations. Q3 enables the data scientist to evaluate which configurations are more likely to generalize well to unseen data, offering a starting point for further tun-

ing or testing. Q4 focuses on selecting the model that is most likely to perform well in real-world scenarios, where the test accuracy serves as a proxy for deployment reliability. Q5 traces back the data that played a role in achieving the best test accuracy, providing a clear link between the input data and model performance. With Q6, data scientists can analyze the relationship between different hyperparameter configurations and DL model performance.

Once a model is deployed, the provenance document becomes an important resource for querying information about its performance and behavior. Post-deployment queries can focus on the reproducibility of the model's performance under different conditions or the impact of any changes made during deployment. These queries provide insights into how the model interacts with new data and whether adjustments to the deployment environment are needed. They also play an important role in model monitoring, troubleshooting, and auditing, ensuring that the model operates as expected. For instance, Q7 can reveal the exact hyperparameters used during training, which is important for understanding how different configurations (*e.g.*, learning rate, or batch size) influenced the model's performance. This knowledge can inform future retraining or tuning efforts. Understanding the computational environment where the model was trained (*e.g.*, GPU specifications, memory, OS, or specific software versions), as in Q8, is important for reproducing the training setup, ensuring consistency across different production environments, and troubleshooting potential environment-related issues that might arise during deployment. Q9 identifies the individual or team responsible for training the model, offering accountability and context when revisiting decisions made during training, such as choices related to hyperparameters or data preprocessing. Q10 provides a detailed provenance trail of all steps leading to the creation of a model. This is important in deployment scenarios where accountability is required, ensuring that all details during model training and validation are transparent and documented. Q10 also ensures that the original processes are well-documented, facilitating efficient reproducibility of results. Q11 and Q12 are particularly beneficial as they provide information about the datasets used to train the model. These queries ensure that the data preparation process is traceable and that the provenance document includes detailed information about any data transformations. In a production setting, they help verify that the model was trained on the intended data and serve as an audit trail for evaluating model performance or addressing any data-related issues. Finally, knowing the feature transformations applied to the training data, as queried in Q12, is important for understanding how preprocessing affected the model's performance. For example, if feature scaling, encoding, or dimensionality reduction was used, tracking these steps ensures that future changes in preprocessing do not unintentionally alter the model's behavior. Q12 also helps identify any discrepancies between the transformations applied to the data during DL model training and those used during deployment, ensuring

consistency in preprocessing steps and minimizing the risk of performance degradation due to mismatched transformations.

4.1.5 Provenance Panel

While analyses are available within the *Analysis Layer* through queries and during training, evaluating the metrics solely through tables generated by SQL queries is not straightforward. There is a demand for supplementary monitoring with graphical resources and flexibility in choosing visualizations (HOHMAN *et al.*, 2019). In our initial endeavor to improve the *Analysis Layer* provided by DLProv , we advocate for the incorporation of graphical visual support, complementing the provenance database. This enhancement allows for an evolutionary analysis of data throughout the training of DL models. Achieving this involves integrating the provenance database with Elasticsearch and Kibana, resulting in the creation of a *Provenance Panel* (SILVA *et al.*, 2021a) (developed in collaboration with Master's student Filipe Silva). Elasticsearch and Kibana were selected due to their inclusion in the *Elastic Stack* [11], offering a suite of free and open tools for ingesting, enriching, storing, analyzing, and visualizing data.

This panel allows data scientists to choose visualizations for key data at each epoch, including epoch number, chosen model, elapsed time, start and end time, accuracy, and loss. By transmitting this data to Elasticsearch, dynamic graphs are automatically generated during execution, such as graphs illustrating average time per epoch, accuracy, and loss trends by epoch, and the comparison between training and validation set accuracy.

The *Provenance Panel*, developed in Python, is designed to run periodically. It selectively transfers data from the *DL Model Provenance* database (in MonetDB) to Elasticsearch, as the intention is not to be a replica of the original database, but a search engine for the main application data. The application uses MonetDB database credentials to extract the latest records from MonetDB, employing an SQL query at predefined time intervals for ingestion into Elasticsearch.

4.2 DLProv Specializations

The DLProv suite includes specialized instances tailored to specific use cases of DL workflows. These specializations are distinguished by their integration with the *Training Service Layer*, using components *Training Library Core* and *Provenance Extractor*. Initially, DLProv was considered to support analyses over DL model training and evaluation in arbitrary execution frameworks (PINA *et al.*, 2021). This section introduces three key instances: DLProv for Keras, DLProv for PINNs, and DLProv for the DL life cycle.

[11] `https://www.elastic.co/elastic-stack/`

4.2.1 DLProv for Keras

KerasProv[12] (PINA *et al.*, 2021) was developed to show that DLProv services can be explicitly invoked from DL scripts or can be embedded into different DL frameworks. Its architecture functions as a provenance plugin for software that executes DL workflows, specifically integrating with the Keras API[13].

KerasProv was first introduced during my master's research and later extended to be compatible with our current provenance model. KerasProv invokes DLProv provenance services directly within the functionalities of the Keras API, providing a Python-based interface for capturing provenance data from Keras DL applications. The core design principle behind KerasProv is to preserve the original Keras structure while enabling the online registration of model configurations, model metrics, and their relationships as provenance data, *i.e.*, DL workflow configuration and DL workflow results. This approach ensures minimal interference with the scientist's workflow while maintaining comprehensive provenance documentation, supporting trust and the quality of DL models.

For KerasProv, the *Provenance Extractor* is embedded directly within the Keras library, eliminating the need for manual instrumentation by the data scientist. Instead, the data scientist selects predefined provenance options, domain data, and hyperparameters to be captured. The *Provenance Extractor* then automatically extracts and stores the values of the DL workflow configuration used in each training iteration. The code snippet in Listing 2 shows how provenance data can be captured for a Keras model.

```
hyperparameter_values = {"OPTIMIZER_NAME": True,
"LEARNING_RATE": True,
"DECAY": False,
"MOMENTUM": False,
"NUM_EPOCHS": True,
"BATCH_SIZE": True,
"NUM_LAYERS": True}

model.provenance(dataflow_tag="KerasProv-example",
                 adaptation=True,
                 hyperparameters = hyperparameter_values)
```

Listing 2: Code snippet showing how to use DLProv for Keras. Adapted from (PINA *et al.*, 2021).

In this example, the *hyperparameter_values* dictionary defines which hyperparameters are captured during training. Each key in the dictionary corresponds to the name of a hy-

[12] https://github.com/dbpina/keras-prov
[13] https://keras.io/

perparameter (*e.g.*, OPTIMIZER_NAME, LEARNING_RATE), while the boolean values indicate whether the respective hyperparameter should be included in the provenance data. The `model.provenance()` function is then invoked to capture this information, using the `dataflow_tag` parameter to assign a unique identifier to the model's dataflow. This minimal addition to the code is sufficient to enable provenance capture, as illustrated in Listing 2, without requiring further instrumentation.

This specialization shows how DLProv can be embedded into DL frameworks to avoid manual instrumentation. However, the current implementation is coupled with a specific version of Keras, which limits its compatibility. Nevertheless, this approach highlights how the Keras API and other frameworks could adopt provenance capture as a built-in plugin, making such capabilities readily available for all users in a compatible representation.

4.2.2 DLProv for PINNs

Since DLProv provides flexibility in terms of arbitrary execution frameworks, we extended the provenance data model and, consequently, *Provenance Extractor*, to capture provenance for PINNs, which have specific model configurations and model metrics and are often trained using specific frameworks like DeepXDE (LU *et al.*, 2021) and SciANN (HAGHIGHAT and JUANES, 2021). This instance, called PINNProv (DE OLIVEIRA *et al.*, 2023), was developed in collaboration with Lyncoln S. de Oliveira as part of his M.Sc. thesis (DE OLIVEIRA, 2023).

To capture provenance when using DeepXDE, a specialized library for PINNs, it is necessary to create a class that inherits from the *deepxde.callbacks.Callback* class. This approach allows PINNProv to leverage DeepXDE methods, which are executed during key steps in the training process of the PINN model. The code example in Listing 3 shows how manual instrumentation is carried out, following the structure outlined in Listing 1. Specifically, the code shows how to capture provenance at key moments during training: when the model training begins (*on_train_begin*), when it ends (*on_train_end*), and after each epoch (*on_epoch_end*).

In addition to traditional hyperparameters and metrics, PINNs have specific metrics that are unique to their architecture. These include loss components such as *WEIGHT_LR*, *WEIGHT_LB*, and *WEIGHT_LD*. The ability to capture these specialized metrics, along with any other relevant data, ensures that the full range of training details can be recorded as provenance. With the use of *callback*, PINNProv can be used with DeepXDE for any PINN specification and any of its *back-ends*.

```
class PINNProv(deepxde.callbacks.Callback):
    def __init__(self):
        (...)
        df = Dataflow(dataflow_tag, ['OPTIMIZER_NAME',
        'LEARNING_RATE', 'EPOCHS', 'BATCH_SIZE', 'LAYERS',
        'WEIGHT_LR', 'WEIGHT_LB', 'WEIGHT_LD'],
        ['epoch', 'time_elapsed', 'LOSS', 'LR_train', 'LB_train',
        'LD_train', 'Q_train_error', 'U_train_error'])
        df.save()
    def on_train_begin(self):
        (...)
        t1 = Task(1, dataflow_tag, exec_tag, "Train")
        tf1_input = DataSet("itrain", [Element([opt_name, l_rate,
        epoch, batch, layers_list, weight_lr, weight_lb, weight_ld)])
        t1.add_dataset(tf1_input)
        t1.begin()
    def on_epoch_end(self):
       tf1_output = DataSet("otrain", [Element([epoch, elapsed_time,
       loss_value, lr_value, lb_value, ld_value, err_q_train,
       err_u_train])])
       t1.add_dataset(tf1_output)
       t1.save()
model.train(..., callbacks=[PINNProv()])
```

Listing 3: Code snippet showing how to use DLProv for PINNs. Adapted from (DE OLIVEIRA *et al.*, 2023).

4.2.3 DLProv for the Deep Learning life cycle

The DLProv suite includes an instance as an integration mechanism for provenance data that enables the capture and merge of information across different stages of a DL workflow. Specifically, the *Provenance Integrator Layer* serves as a bridge that facilitates the integration of provenance data, whether it originates from preprocessing, model training, or other steps within the DL workflow. While the *Provenance Integrator Layer* is designed to be adaptable and extendable to accommodate different provenance capture solutions, its current implementation focuses on integrating data from two key sources: preprocessing steps, as captured by Chapman *et al.* (CHAPMAN *et al.*, 2020), and DL model training and selection, as captured by DLProv.

Chapman *et al.* (CHAPMAN *et al.*, 2020) proposed a provenance capture tool for ML preprocessing, wherein provenance is captured when operators are applied to datasets. They also proposed a formalization for the set of operations, such as data reduction, augmentation, and transformation. The main purpose of the provenance capture in ML data preprocessing is to allow the data scientist to analyze, understand, and debug what hap-

pened in the execution. As the provenance is captured in a very fine grain, it allows the data scientist to track individual data items. The tool is backed by a MongoDB[14] database that is used as a provenance store. The implemented system was tested over real-world ML benchmark pipelines like German Credit, Compas Score, and Census, and the results showed that it is possible to collect fine-grained provenance that is helpful for the data scientist's analysis, such as analyzing why applying an imputation during preprocessing on train/test dataset, the same values are predicted for all rows.

In the preprocessing step, provenance data is collected by a dedicated module that tracks transformations and modifications made to the raw data. This could include operations such as normalization, feature extraction, and data augmentation. The provenance information gathered in this step includes not only the transformations applied to the data but also metadata such as the parameters used in each operation, the timestamp of when the transformations occurred, and the resulting data artifacts. By capturing this information, a solution ensures that the data preparation step is fully traceable, which is essential for reproducibility and transparency in DL workflows. Once the data is prepared, the training process begins. A data scientist can leverage DLProv to capture provenance throughout the DL model's training.

The *Provenance Integrator Layer* consolidates this diverse provenance data into a unified structure that can be queried, visualized, and analyzed. This integration is important for supporting complex analyses, such as identifying correlations between data transformations and model performance, detecting issues related to data leakage, or ensuring that the same preprocessing steps used during training are consistently applied when the model is deployed in production.

To enable the integration of provenance data models from two provenance capture tools, the first step is to obtain a unique identifier for each data record (*e.g.*, record_id), which can be retrieved by querying the *Data Preparation Provenance* database, responsible for storing the provenance data for the preprocessing operations. Once the record_id is obtained, it is inserted into the *DL Model Provenance* database (in MonetDB), along with the type of data transformation associated with that record (*i.e.*, whether the record will be used for training or evaluation of a DL model). Since these records play an important role in the model development and selection process of the DL workflow, and MonetDB serves as the provenance database during this stage, the record_id attribute must be present in both provenance models to establish a foreign key relationship between the two databases.

Figure 4.4 shows on the left a fragment of the dataset Framingham Heart Disease (FHS), with features male, age, education, cigsPerDay, and glucose. Each record has a unique identifier called record_id. Since the provenance for the preprocessing is collected at an

[14] `https://www.mongodb.com/`

attribute level, each data value receives an identifier. Examples of the record_id and identifier can be seen in Figure 4.5. On the right of Figure 4.4, some preprocessing operations are applied to the data, such as removing the feature *education*. As a result of this operator, the tool generates a provenance document such as the one shown on the right of Figure 4.5, where there is an activity named *Dimensionality reduction* applied to the feature called *education*. This activity was responsible for invalidating (*i.e.* removing) the feature education for each record_id, generating a relation of the type *wasInvalidatedBy*, containing all data values for the education feature.

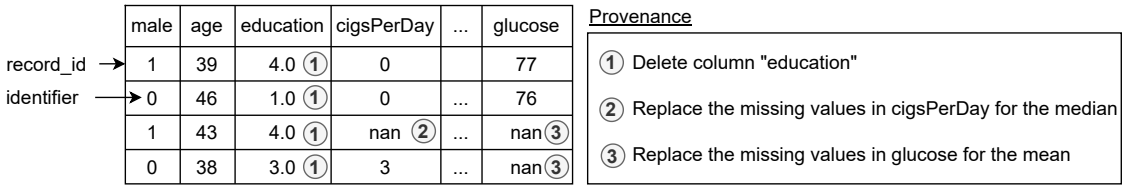


Figure 4.4: Current provenance model granularity for preprocessing.

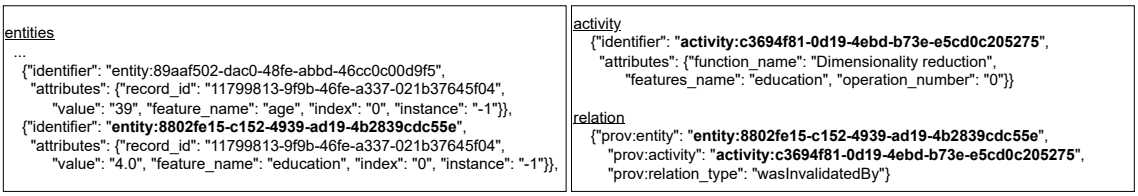


Figure 4.5: Example of provenance captured during the preprocessing step.

After preparing the data to train a DL model, the data scientist needs to define the hyperparameters' values and the DL architecture. During this training/evaluation, it is important to analyze the metrics resulting from these configurations. Figure 4.6 shows two configurations to train a DL model and the metrics (*i.e.* loss and accuracy) achieved during the training of these two sets of hyperparameters. Training consumes (*used*) a series of hyperparameter values such as the name of the optimizer, learning rate, number of epochs, and number of layers in the network, and produces (*wasGeneratedBy*) a set of metrics, such as the accuracy, the value of the loss function, and the elapsed time of each epoch. Relations *used* and *wasGeneratedBy* are represented in the provenance document shown in Figure 4.7. Note that the DLProv's provenance is represented at a row level with the provenance data model, so it considers a set of hyperparameters as an entity, as well as a set of metrics for each epoch in the training.

Figure 4.8 presents a fragment of our provenance model, represented as a UML class diagram, along with a fragment of the instantiation of the preprocessing provenance model for the running example. It illustrates the origin of the *record_id*, which is used to integrate these two provenance documents.

	learning rate	optimizer	num epochs	batch size	...	num layers
①	0.001	adam	1000	64	...	33
①	0.0005	sgd	20000	32	...	32

	epoch	elaspsed time	loss	validation loss	accuracy	validation accuracy
②	1	2.05	26.88	16.53	0.84	0.84
②	2	0.48	7.56	1.82	0.81	0.77
②	3	0.50	1.52	1.42	0.73	0.76
②	1	1.43	1.20	0.45	0.63	0.84
②	2	0.21	0.44	0.45	0.85	0.84
②	3	0.34	0.44	0.44	0.85	0.84

Provenance

① Used to train the DNN

② Metrics values achieved during training

Figure 4.6: Current provenance model granularity for model selection.

```
prefix dnnprov <DNNProv>
entity(dnnprov:i<b>Training01</b>, [dnnprov:learning_rate="0.001",
dnnprov:optimizer="adam", dnnprov:num_epochs=1000,
dnnprov:batch_size=64, dnnprov:num_layers=33])
entity(dnnprov:<b>oTraining_07</b>, [dnnprov:epoch=1, dnnprov:elapsed_time="2.05",
dnnprov:loss="26.88", dnnprov:validation_loss="16.53",
dnnprov:accuracy="0.84", dnnprov:validation_accuracy="0.84"])
entity(dnnprov:<b>oTraining_08</b>, [dnnprov:epoch=2, dnnprov:elapsed_time="0.48",
dnnprov:loss="7.56", dnnprov:validation_loss="1.82", dnnprov:accuracy="0.81",
dnnprov:validation_accuracy="0.77"])
entity(dnnprov:<b>oTraining_09</b>, [dnnprov:epoch=3, dnnprov:elapsed_time="0.5"
%% xsd:float, dnnprov:loss="1.52", dnnprov:validation_loss="1.42",
dnnprov:accuracy="0.73", dnnprov:validation_accuracy="0.76"])
activity(keras-prov:<b>TrainingDNN_25</b>, -, -)
wasGeneratedBy(dnnprov:<b>oTraining_07</b>, dnnprov:<b>TrainingDNN_25</b>, -)
wasGeneratedBy(dnnprov:<b>oTraining_08</b>, dnnprov:<b>TrainingDNN_25</b>, -)
wasGeneratedBy(dnnprov:<b>oTraining_09</b>, dnnprov:<b>TrainingDNN_25</b>, -)
used(dnnprov:TrainingDNN_25, dnnprov:<b>iTraining01</b>, -)
```

Figure 4.7: Example of provenance in the model selection.

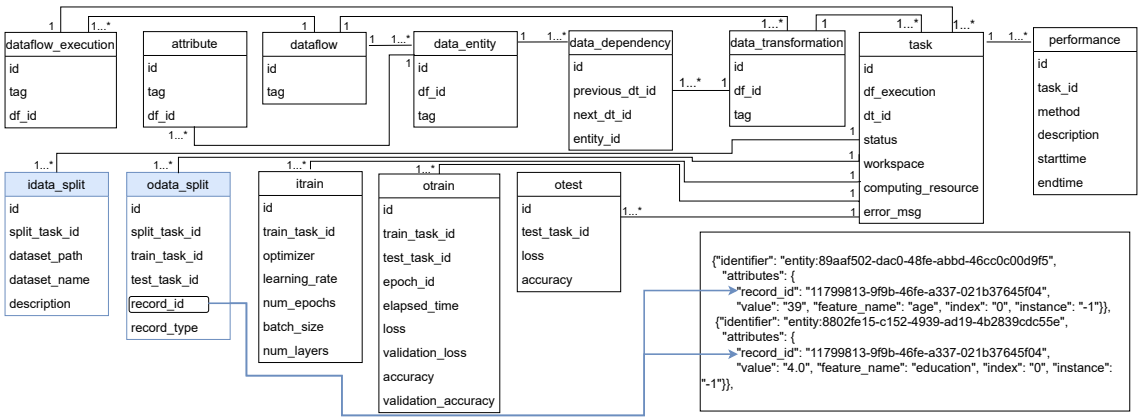


Figure 4.8: DLProv provenance data model and instantiation of the preprocessing provenance.

Figure 4.9 shows the steps of the proposed integration between these two provenance capture tools to allow the integration between the data models of the two provenance capture tools, first, it is necessary to obtain the identifier for each record (record_id), and this can be performed by querying the *Data Preparation Provenance* database. At this time, the *Provenance Integrator Layer* considers only structured data (*e.g.* tabular data). This step is called *mapping*. We propose three mapping alternatives, which will be further described. Second, the obtained record_id is inserted in the *DL Model Provenance* database (in MonetDB), along with the type of data transformation that it is going to use that record (*i.e.* if the record is going to be used to train a DL model or test/evaluate). Since the records are used in the model selection process of the DL workflow and MonetDB is the provenance database for this step, the record_id attribute must be in both models to act as a foreign key between provenance databases.

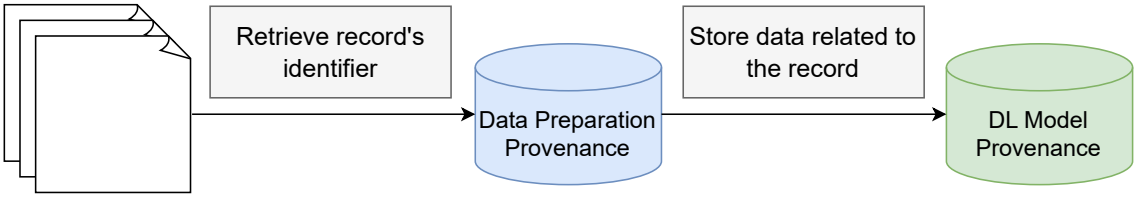


Figure 4.9: Diagram for the mapping. Adapted from (PINA *et al.*, 2023).

We present three mapping alternatives. The first is straightforward. When splitting the dataset, the mapping function identifies the record_id through a MongoDB query that returns the attributes and values of each record, as shown in Listing 4.

```
for each record in dataset:
    cursor = db.entities.find(
    {"attributes.feature_name": attribute, "attributes.value": value})
    return record_id
```

Listing 4: Simple mapping alternative for retrieving record_id from MongoDB.

Due to the granularity of the preprocessing provenance (attribute level), this query would have to be processed for each attribute in a record, and then find the common record_id. However, the processing time for each row is too long. For instance, the execution of this query took approximately 996.17 seconds for 1,000 rows of the Framingham dataset (which contains 4,240 rows). Therefore, this alternative did not achieve good results. The second alternative is to add to the preprocessing provenance capture the mapping between the record and the record_id. The idea is to write the entities along with the mapping between record_id and attributes as presented in Listing 5.

As the final dataset is composed of the preprocessed data, every time a data value is modified or deleted during the preprocessing step, the mapping should be updated to

```
for each record in dataset:
    attributes = {}
    attributes['record_id'] = record_id
    for i in record.keys(): attributes[i] = record[i]
```

Listing 5: Mapping alternative for the *record_id* during preprocessing provenance capture.

reflect that, thus finding the correct record_id. At the end of the preprocessing step, the resulting JSON for the mapping is persisted in MongoDB. With this alternative, only one query would be necessary to identify the record_id of a specific record, as presented in Listing 6.

```
cursor = db.mapping_records.find(
{'$and':{"male": 1},{"age": 39},{"currentSmoker": 0},{"cigsPerDay": 0},
{"BPMeds": 0.0},{"prevalentStroke": 0},{"prevalentHyp": 0},
{"diabetes": 0},{"totChol": 195},{"sysBP": 106},{"diaBP": 70},
{"BMI": 26.97},{"heartRate": 80},{"glucose": 77}})
```

Listing 6: Example illustrating how to retrieve the *record_id* of a tuple.

For the second alternative, the average processing time for the queries and inserting the results into MonetDB (10 repetitions) for the Framingham dataset was 49.85 seconds, 205.8 seconds for 1,000 records of the Adult Census dataset, and 1,496.13 seconds for the Credit Card Fraud dataset. Therefore, it is still taking a long time to process the queries.

The third alternative is an improvement of the second one. Instead of writing the JSON with the mapping between record_id, attributes, and values, a hash function (hashlib[15]) is applied to the dictionary corresponding to each record, and then, the mapping is in the format presented in Listing 7.

```
{"record_id": id_value, "hash": hash_value}
```

Listing 7: Mapping alternative for the *record_id* using a hash function.

The hash function is applied to the dictionary of each record without containing the record_id. Therefore, to find the record_id for a specific record, a simple query can get the record_id that matches a hash value. This process is depicted in Figure 4.10. The average

[15] https://docs.python.org/3/library/hashlib.html

time to apply the hash function to each record, process the query to obtain the record_id, and insert the results into MonetDB was 14.93 seconds for the Framingham dataset, 94.8 seconds for the Adult Census, and 139.41 seconds for the Credit Card Fraud. With these results, we chose this mapping alternative in our experiments.

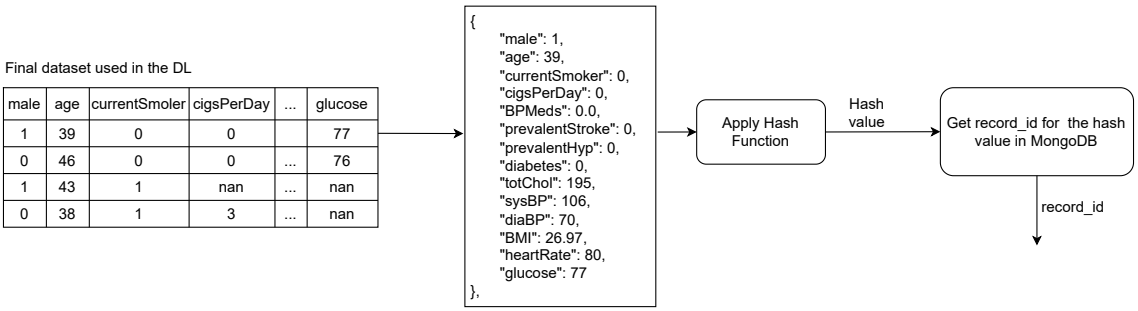


Figure 4.10: Example of the process to get the record_id.

This mapping adds an overhead to the execution of a DL model script due to obtaining the record_id of each record and inserting it into MonetDB each time the dataset is split for training a DL model. To mitigate, or at least reduce this overhead, the use of multi-threading [16] was introduced. In the DL model training, the step of obtaining the record_id, and saving it in the database responsible for managing the DL model training data occurs simultaneously, reducing the processing impact of this step.

[16] https://docs.python.org/3/library/threading.html

Chapter 5

DLProv in Action: Traceability for Analyses of Deep Learning Workflows

This chapter presents the experimental evaluation of DLProv, highlighting its ability to integrate data derivation traces fragmented across different workflow steps, represent provenance data, operate independently of specific DL frameworks and computational environments, and generate the corresponding provenance graph.

Table 5.1 provides an overview of the experiments presented in this chapter, covering a diverse set of datasets, computational environments, and DL frameworks. The *Dataset* column identifies the specific dataset used in each experiment, covering a wide range of domains including healthcare, finance, computer vision, scientific simulations, and handwritten document analysis. The *Dataset Characteristics* column classifies the type of data used, such as text, image, numeric, scientific, or handwritten images, reflecting the diversity in data. The *Computational Environment* column specifies the execution environments used in each case, ranging from the use of CPUs in personal laptops and GPUs in Google Colab to high-performance computing environments. The *DL Framework* column identifies the DL framework used in the experiment, including TensorFlow, PyTorch, and DeepXDE, thereby highlighting DLProv's independence of DL frameworks. Lastly, the *RQ* column indicates which Research Question each experiment addresses. While most of these experiments use Python scripts, some adopt Parsl to facilitate workflow parallelization and resource management.

Table 5.1: Overview of the experiments conducted to evaluate DLProv.

Dataset	Dataset Characteristics	Computational Environment	DL Framework	RQ
Framingham Heart Disease	Text	4 CPU cores	TensorFlow	RQ1, RQ4
Adult Census Income	Text	4 CPU cores	TensorFlow	RQ1
Credit Card Fraud Detection	Text	4 CPU cores	TensorFlow	RQ1
Iris	Text	12 CPU cores	TensorFlow	RQ2
Oxford Flowers	Image	12 CPU cores	TensorFlow	RQ2
Oxford Flowers	Image	1 GPU	TensorFlow	RQ3
ImageNet	Image	1 GPU	TensorFlow	RQ3
MNIST	Image	2 GPUs	PyTorch	RQ3
CIFAR-100	Image	2 GPUs	TensorFlow	RQ3
Brazilian Forensic Letter	Handwritten Images	2 GPUs	PyTorch	RQ3
Velocity fields and Seimic images	Scientific	4 CPU nodes	TensorFlow	RQ3
Seismic images	Scientific	2 GPUs	TensorFlow	RQ3
Seismic images	Scientific	1 GPU	TensorFlow	RQ3
$x \in [-1, 1]$	Numeric	1 GPU	DeepXDE	RQ3

5.1 Integrating Provenance Traces for Deep Learning Workflow Analyses

This experiment focuses on RQ1 by showing the integration of data derivation traces fragmented across different DL workflow steps, such as data preparation and DL model training, to provide traceability. We evaluate real DL workflows using the proposed integration approach from Section 4.2.3, combining DL training, evaluation, and preprocessing steps to ensure traceability. This integration is complex as it involves: *(i)* two distinct provenance traces, *(ii)* tuple-level granularity, with even finer granularity for data preparation, and *(iii)* validation through Apache Drill[1], integrating MongoDB and MonetDB. This experiment serves as a proof of concept, showing the feasibility of integrating data preparation (*i.e.* provenance of preprocessing operations applied to the input dataset) with the training of a DL model, following the proposed provenance representation.

[1] `https://drill.apache.org`

Table 4.1 presents a series of provenance queries that are commonly submitted to provenance databases. To evaluate if the proposed integration approach can answer these queries, we have captured provenance data in three real-world datasets involving different types of preprocessing steps. Then, the resulting data from these steps is used to train a DL model. Following, we describe the datasets and preprocessing operations used in the experiments.

The Framingham Heart Disease (FHS)[2] dataset has over 4,000 records and 15 attributes, such as gender, whether a person is a smoker, age, level of education, and how many cigarettes are smoked per day. The target of the dataset is to predict the 10-year risk of coronary heart disease (CHD). The goal of the Adult Census Income[3] experiment is to predict whether annual income for an individual exceeds $50K based on census data. These data were extracted from the 1994 Census Bureau database, which has over 32,000 records and 15 attributes, such as age, education, marital status, occupation, race, sex, capital gain and loss, and native country. Finally, the Credit Card Fraud Detection[4] dataset contains transactions made by credit cards in September 2013 by European cardholders, with over 284,000 records. For the experiments presented in this section, we consider 50,000 records out of the 284,000. As a security concern, the actual variables are not shared, but they have been transformed using principal component analysis (PCA). As a result, we can find 30 feature columns and 1 final class column. The goal of this experiment is to help credit card companies identify fraudulent transactions so that customers are not charged for items that they did not purchase. Table 5.2 shows the preprocessing steps for each of these datasets.

The experiments were performed using TensorFlow[5] on a laptop with Intel Core i7-8565U 1.80GHz and 16GB RAM running Windows 10. They aim to show the analytical potential of the proposed approach rather than evaluating the performance of a trained model. Common layers, such as dense layers with activations like ReLU and Sigmoid, were employed.

To reiterate, this integration involves provenance traces from two distinct solutions, one from (CHAPMAN *et al.*, 2020) and the other from DLProv. Provenance from preprocessing is stored in MongoDB, while provenance from DL model training and evaluation is stored in MonetDB, requiring combined queries to reconstruct complete workflows. Moreover, provenance data in this scenario varies in granularity. Training provenance in MonetDB is structured at the tuple level, whereas preprocessing provenance in MongoDB is stored as nested JSON at the attribute level. Our provenance model ensures that

[2] https://www.kaggle.com/datasets/aasheesh200/framingham-heart-study-dataset
[3] https://www.kaggle.com/datasets/uciml/adult-census-income
[4] https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
[5] https://www.tensorflow.org/

Table 5.2: Operations for the DL workflow.

Dataset	Id	Description
Framingham Heart Disease	A0	education column was deleted.
	A1	Replace the missing value in cigsPerDay for the median.
	A2	Replace BPMeds missing values for 0 or 1.
	A3	Replace NaN in totChol, BMI, and glucose for the mean.
	A4	Replace NaN in heartRate using bfill.
	A5	TenYearCHD column was deleted.
	A6	TenYearCHD was defined as the target column.
	A7	Split dataset into train and test.
	A8	Define DL architecture layers.
	A9	Define hyperparameters and metrics.
Adult Census Income	B0	Remove whitespace from 9 columns.
	B1	Replace '?' character for NaN value.
	B2	7 categorical columns were OneHot encoded.
	B3	Two columns were binarized.
	B4	fnlwgt column was deleted.
	B5	label column was deleted.
	B6	label was defined as the target column.
	B7	Split dataset into train and test.
	B8	Define DL architecture layers.
	B9	Define hyperparameters and metrics.
Credit Card Fraud Detection	C0	Select 50,000 rows.
	C1	Time column was deleted.
	C2	Amount column was scaled.
	C3	Duplicate rows were deleted.
	C4	Class column was deleted.
	C5	Class was defined as the target column.
	C6	Split dataset into train and test.
	C7	Define DL architecture layers.
	C8	Define hyperparameters and metrics.

queries operate at the appropriate granularity for each data source, enabling fine-grained provenance integration and preserving the richness of both representations. Apache Drill facilitates this integration by enabling direct SQL queries across both databases.

To achieve this, Drill requires configuring storage plugins to connect to MongoDB and MonetDB. Each plugin defines the connection parameters, including authentication details, endpoints, and specific settings for handling JSON structures in MongoDB or relational tables in MonetDB. Once configured, scientists can write SQL queries that seamlessly join data across these sources, treating them as part of a unified queryable namespace. With the storage plugins configured, several provenance queries were submitted to validate the integration and retrieve insights from both MongoDB and MonetDB.

Provenance Queries Analysis

Considering the complete workflow presented in Table 5.2 (preprocessing and model selection steps), the data scientist applies operations such as removing the whitespace from a few columns, replacing '?' with NaN, and deleting other columns from the Census dataset. The next step in the workflow, after preparing the data, is splitting the data into the Train and Test datasets. Then, the data scientist defines the DL architecture, *i.e.* the number and type of layers of the DL model, their activation functions, the hyperparameters that are going to be used to train the DL model (for example, learning rate and optimizer), and the metrics used to evaluate the trained model. As previously mentioned, DL model selection steps involve exploratory analysis as well as tuning hyperparameters to improve the accuracy of the trained model. Thus, this process of defining the DL workflow configuration and analyzing the DL workflow results can happen multiple times, until the "best" model, according to the criteria defined by the data scientist, is chosen.

We trained the DL model on the Census dataset, varying hyperparameters such as the learning rate with values 0.001 and 0.002, the optimizers Adam and SGD, and the number of epochs 100 and 200. By submitting queries like Q3 and Q6, we identified the best results: a loss of 0.32 and an accuracy of 0.86 on the test set. These were achieved with a DL workflow configuration that included a learning rate of 0.001, 200 epochs, a batch size of 32, and the Adam optimizer. Such queries enable the association of DL workflow configurations, including hyperparameters, with model performance metrics.

As our provenance data model (Figure 4.3) considers data such as the dataset name, description, and path, the answer to Q11 can be in a more general format, such as "The dataset used was Adult Census, obtained from https://www.kaggle.com/datasets/uciml/adult-census-income", or something more specific, such as the records that were used to train and evaluate a model. So, the answer to this query can be with the records that were consumed for DL training. For the modified Q11, which considers the data used to evaluate a trained model, the answer would include only the records involved in the DL model evaluation, in other words, the test set. The records in each dataset, training, and testing, may be different between one training run and another. Therefore, to answer these queries, it is possible to make associations in our approach on account of provenance (data derivation).

For DL training for the Framingham dataset, initially, records with missing values were removed, and records associated with people younger than 50 were deleted. The second proposed preprocessing followed the operations presented in Table 5.2. Training and testing were performed considering the datasets resulting from these two sequences of preprocessing operations, with different sets of hyperparameters. Through the analysis of the query Q12, it was possible to verify that the first proposed sequence produced

unsatisfactory results (loss = 0.52 and accuracy = 0.83). This analysis is possible because we have registered through provenance the operations that were applied to the dataset in the preprocessing provenance database, and the hyperparameters, the metrics, and the mapping proposed by our integration approach for each iteration in the DL workflow.

The operators applied to the Credit Card Fraud dataset are presented in Table 5.2, and the DL model was trained with its resulting dataset. Considering that this is a heavily unbalanced dataset due to the ratio of fraud to non-fraud transactions (fraud cases are only 492 of 284,807 transactions), and 50,000 records were used for this example, two approaches were used for Op C0. The first was to select the initial 50,000 records (which led to 140 fraud cases and 48,746 non-fraud transactions), and the second was to use *oversampling* to duplicate the minority class records to be equal to that of the majority class. Therefore, while the first approach is a data reduction, the second is a data reduction followed by data augmentation. Both resulting datasets were trained with different configurations of hyperparameters. Using our proposed approach, we observed that training the DL model with the dataset from the first approach obtained values for loss and accuracy in the test set for 100 epochs around 0.007 and 0.99, respectively, and 0.011 and 0.99 for 200 epochs, as the question in Q7. We can also see that the metrics obtained during training presented the same behavior. Training with the dataset from the second approach generated a loss value of 4.17 and an accuracy of 0.95 in the evaluation of the model using the test set for 100 epochs, and a loss value of 2.95 and an accuracy of 0.95 for 200 epochs. Then, with the records used to train and test, we can trace back these results to the preprocessing steps applied to the data (Q12) and verify their differences to make the best decision as to which preprocessing steps and final model to deploy or to have insights for configuring future trials.

With the proposed integration approach, it is also possible to analyze use cases Q13 and Q14. In Q13, the data scientist can investigate high loss values even after tuning hyperparameters and applying data augmentation by associating the recorded loss values with the corresponding hyperparameters and tracing back to the preprocessing steps applied to the data. Similarly, in Q14, where training on an imbalanced dataset results in poor validation accuracy, the scientist can trace back to the dataset composition and preprocessing choices to assess their impact on model performance.

Overhead Analysis

The overhead introduced by provenance capture with DLProv has been evaluated in published papers, being less than 3% in High-Performance Computing environments (PINA *et al.*, 2021; SILVA *et al.*, 2021b). In the experiments presented in this section, we report the overhead introduced to the DL training process with the improved *Data Layer* and *Provenance Integrator Layer*, considering provenance capture with the mapping for

the datasets used to train and test the DL model. Several training runs were performed for the datasets, with variations in hyperparameter values, *e.g.* number of epochs (100, 200, 300), learning rate (0.001, 0.002), and batch size (16, 32). Considering this process without using multithreading, we noticed that the overhead was above 50%. However, in faster training (using less than 100 epochs), this value reached up to 130%.

On the other hand, when the experiments were performed using multithreading, we observed that very fast training processes (using less than 100 epochs) presented an acceptable overhead that fluctuates from 17% to 32%, for the datasets used in these experiments. Furthermore, we noticed that the larger the dataset, the smaller this overhead is. For longer executions, using at least 200 epochs in our experiments, the overhead was less than 10%. Therefore, this overhead can be considered acceptable, especially in these cases, considering that the data scientist can have the benefit of processing queries to help understand and fine-tune the DL workflow.

5.2 DL workflows traceability analysis through provenance graphs

The experiments in this section focus on DL workflows analysis through provenance graphs, taking advantage of how provenance data is represented with W3C PROV, addressing RQ2. We initially evaluate the support of Weights & Biases (WandB) and its limitations for provenance analyses. Then, we present a practical evaluation of the analytical traceability provided by DLProv, focusing on query-based analyses. The experiments were performed on a laptop with an Apple M2 Pro chip and 16GB of RAM running macOS.

5.2.1 Traceability in Weights & Biases

Weights & Biases (WandB) allows the tracking and versioning of data as the inputs and outputs of DL workflow executions[6]. It is possible to log hyperparameters, metadata, and metrics, and use an artifact to log, track, and version the dataset used to train the model as input, and another artifact for the resulting model checkpoints as output. WandB provides a lineage map, as shown in Figure 5.1, following these logged and used artifacts.

Using a simple DL model with the Iris dataset[7], we conducted experiments while leveraging WandB for experiment tracking. Throughout this process, we logged key artifacts including the original dataset, the preprocessed data, and the configuration parameters

[6] `https://docs.wandb.ai/guides/artifacts/`
[7] `https://archive.ics.uci.edu/dataset/53/iris`

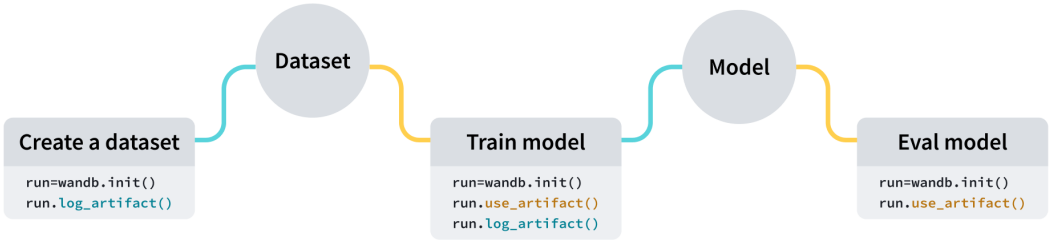


Figure 5.1: Weights & Biases lineage map. From `https://docs.wandb.ai/guides/artifacts/`.

used during DL model training. Figure 5.2 shows the resulting lineage map, automatically generated by WandB, which captures the dependencies among these artifacts.

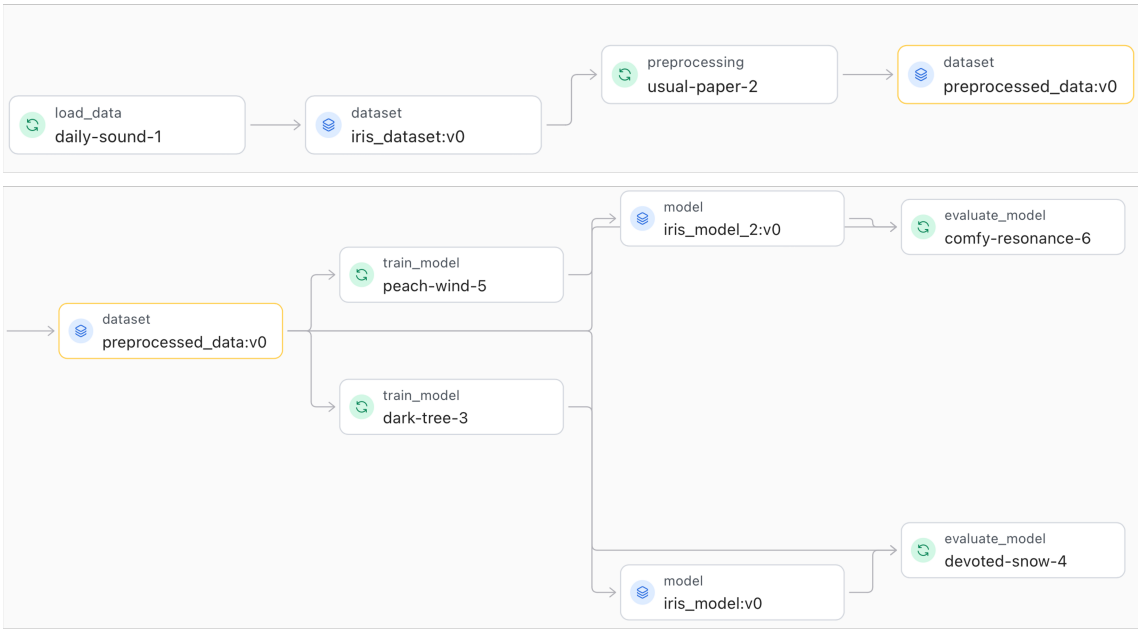


Figure 5.2: Lineage map generated using Weights & Biases (WandB) during experiments with AlexNet.

WandB offers methods to retrieve logged and used artifacts. The method `artifact.logged_by()` returns the run that originally created or logged a given artifact, while `artifact.used_by()` returns a list of runs that have used that artifact as an input. On the other hand, from the perspective of a specific run, `run.logged_artifacts()` retrieves all artifacts that were output or logged by that run. Meanwhile, `run.used_artifacts()` returns only the input artifacts that were explicitly declared as dependencies via `run.use_artifact()`.

To answer queries such as *Find all trained models with a learning rate of 0.002. What feature transformations have been applied to the data?*, we traversed the artifact lineage using the WandB API methods described above. Listings 8 and 9 show how these queries were implemented.

```python
import wandb
api = wandb.Api()
runs_from_project = api.runs(path="dlprov/dl-workflow",
filters={"config.learning_rate": 0.002})
for run in runs_from_project:
    print("Run ID:", run.id)
    for key, value in run.config.items():
        print(f"{key}: {value}")
```

Listing 8: Query "Find all trained models with a learning rate of 0.002".

```python
import wandb
api = wandb.Api()
runs_from_project = api.runs(path="dlprov/dl-workflow",
filters={"jobType": "train_model"})

for run in runs_from_project:
    consumed_artifacts = run.used_artifacts()
    for p in consumed_artifacts:
        if (p.type == "dataset"):
            print("The dataset used to train the produced model was",
            p.name)
            run_that_logged = p.logged_by()
            print("This dataset was logged in the run with id ",
            run_that_logged.id)
            print("The run was of the type ", run_that_logged.jobType)
            original_dataset_run = run_that_logged.used_artifacts()
            for r in original_dataset_run:
                if (r.type == "dataset"):
                    print("The original dataset was", r.name)
                    run_that_logged_dataset = r.logged_by()
                    print("This dataset was logged by",
                    run_that_logged_dataset.id)
                    print("This run was identified as",
                    run_that_logged_dataset.jobType)
                    print("##########")
```

Listing 9: Query "What feature transformations have been applied to the data?"

Queries involving specific entities, such as identifying which runs used a specific hyperparameter, can be easily answered using the WandB interface or a few lines of script. However, queries that involve relationships between artifacts, such as tracing the sequence of transformations or understanding how a model was derived from a particular dataset,

require a more involved approach. These relationship-based queries often require traversing the lineage map through code, as the provenance information must be pieced together by following links between input and output artifacts across runs.

5.2.2 Comparing DLProv with MLflow and MLflow2PROV

In this section, we compare DLProv's ability to answer queries with that of MLflow and MLflow2PROV. MLflow was selected because it is a widely adopted tool for managing ML experiments. MLflow2PROV was included as it extends MLflow by converting tracked information into W3C PROV format, providing a relevant baseline for evaluating provenance-based analytical capabilities.

To provide this comparison, we trained a DL model following the AlexNet DL architecture (KRIZHEVSKY *et al.*, 2012) using the Oxford Flowers (NILSBACK and ZISSERMAN, 2006) to extract provenance graphs from these experiments. AlexNet is a pioneering convolutional neural network (CNN) for image classification, especially notable for its performance with high-resolution images. Designed to categorize images across more than 1,000 classes, AlexNet accepts an image input size of 227 *x* 227 pixels. The architecture, shown in Figure 5.3, includes eight layers: five convolutional layers followed by three fully connected layers. AlexNet uses Rectified Linear Units (ReLU) as its activation function instead of the hyperbolic tangent (tanh) function, which was standard at the time, and this reduced the training time.

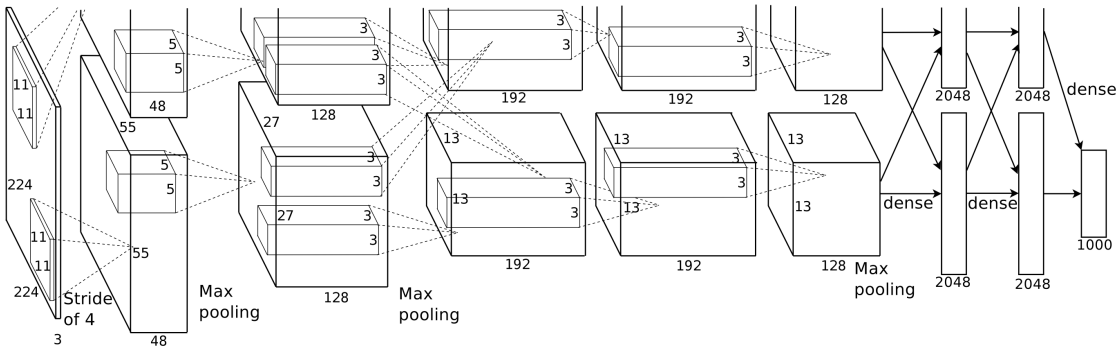


Figure 5.3: AlexNet architecture (KRIZHEVSKY *et al.*, 2012).

In our experiments, we employ AlexNet using the Oxford Flower dataset, which includes 17 species of flowers with 80 images per class. The Oxford Flower dataset presents challenges due to its inclusion of visually ambiguous categories: some classes cannot be differentiated solely by color (*e.g.*, dandelions vs. buttercups), while others are difficult to distinguish by shape alone (*e.g.*, daffodils vs. windflowers).

The same DL workflow was executed using MLflow and DLProv. Since MLflow does not provide provenance data relationships, we used the MLflow2PROV solution to execute

queries on provenance traces. Provenance-centric approaches explicitly capture relationships in the workflow, showing how each step in the workflow is connected. This enables scientists to trace back through the entire workflow, understand dependencies, and make more informed decisions based on the complete history of their experiments.

MLflow logs data during execution, including metadata about the datasets and model metrics, storing the captured data within a directory. Through the MLflow user interface (UI), scientists can access an overview of executions and experiments, along with key information and metrics. However, MLflow does not capture preprocessing operations or relationships like *wasGenerateBy* and *used* that associate datasets with activities, relying on the scientist to log such information and write programs to identify and trace them (GRAFBERGER *et al.*, 2022). While MLflow can address entity-level queries regarding hyperparameters, metrics, and metadata, it faces limitations in answering relationship-level queries, such as the ones that need associations relating to preprocessing, model training, and evaluation. Queries involving metrics averaging, minimum, or maximum values cannot be directly answered within the MLflow UI and may require external procedures.

The MLflow client can also use an SQLAlchemy-compatible database (*e.g.*, SQLite, PostgreSQL, MySQL) for the backend. When using a database backend, metadata is stored in the database, and artifacts are stored under a local *./mlruns* directory. This setup allows scientists to submit queries to the database, besides using the MLflow UI, and queries involving metrics averaging, minimum, or maximum values can be answered. However, some of the logged data are stored only in the MLflow artifacts directory (*i.e.*, not in the DBMS), and as a result, these log files may be separated from the model metadata tracked in the DBMS, requiring post-processing for data access. MLflow2PROV extracts provenance information from ML experiments conducted using MLflow and storing data in Git repositories. By implementing a W3C PROV-compliant provenance model, MLflow2PROV enables the generation of provenance graphs that capture activities within ML development and experiment projects.

To capture metrics, parameters, and artifacts, MLflow and DLProv require script instrumentation on the scientists' part. Figure 5.4 depicts a fragment of a provenance graph generated by DLProv's provenance graph generation functionality, following the proposed provenance representation. DLProv enables query submission during training, allowing answering questions such as what artifacts were used as input to the activity *Testing_101*. For example, considering the activity *Testing_101* depicted in Figure 5.4, DLProv's response for the aforementioned query would generate a provenance graph representing that this activity *used* the entities *Testing_101*, *dlprov:170.229.25.114*, and *oTestingTuples_-101*. In contrast, since MLflow relies only on metadata, without establishing relationships

between data (*i.e.*, entities) and data transformations (*i.e.*, activities), a query of this type cannot be accomplished within the MLflow platform.

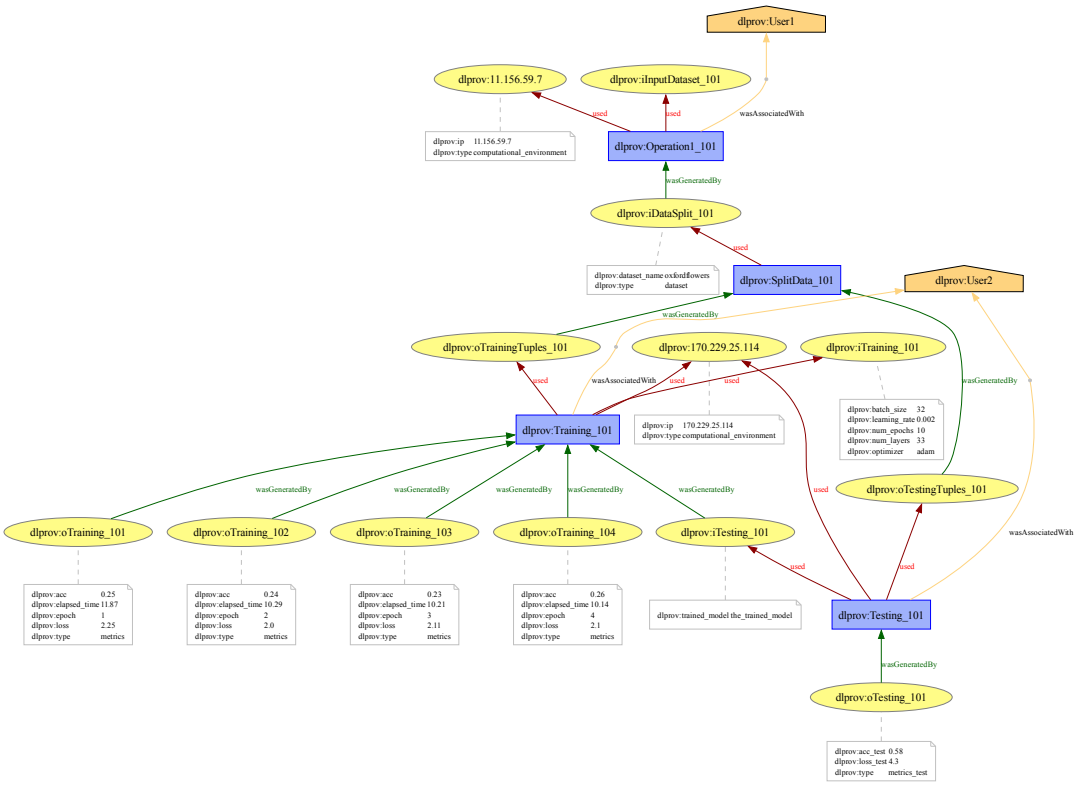


Figure 5.4: W3C provenance graph fragment for a single AlexNet model training.

To provide a fair comparison (since MLflow2PROV uses a G-DBMS to represent provenance), DLProv uses Prov Python and PROV Database Connector[8] to generate the graph in DLProv and insert it in a G-DBMS. Thus, the AlexNet provenance database was mapped to Neo4j as shown in Figure 5.5. Figure 5.6 presents the provenance graph in Neo4j generated by MLflow2PROV. Different from DLProv, which stores DL workflow activities executions in the provenance database, MLflow2PROV provenance stores coarse-grained activities, *e.g.*, RunCreation that represents an execution of the workflow, and multiple entities that represent the artifacts, parameters, and metrics. Such entities are associated using a *hadMember* relationship. In addition, MLflow2PROV incorporates versioning entities in the same provenance graph, *e.g.,* considering Git commits.

We evaluated the three approaches in answering queries presented in Table 4.1. We consider that the tool supports a query when its provenance database allows for obtaining the query result through the relationships, *e.g.*, using a specific query language, or its interface allows for obtaining the result. Table 5.3 shows if each approach addresses the queries. A check symbol (✓) indicates that the approach answers the query, while

[8] `https://github.com/DLR-SC/prov-db-connector`

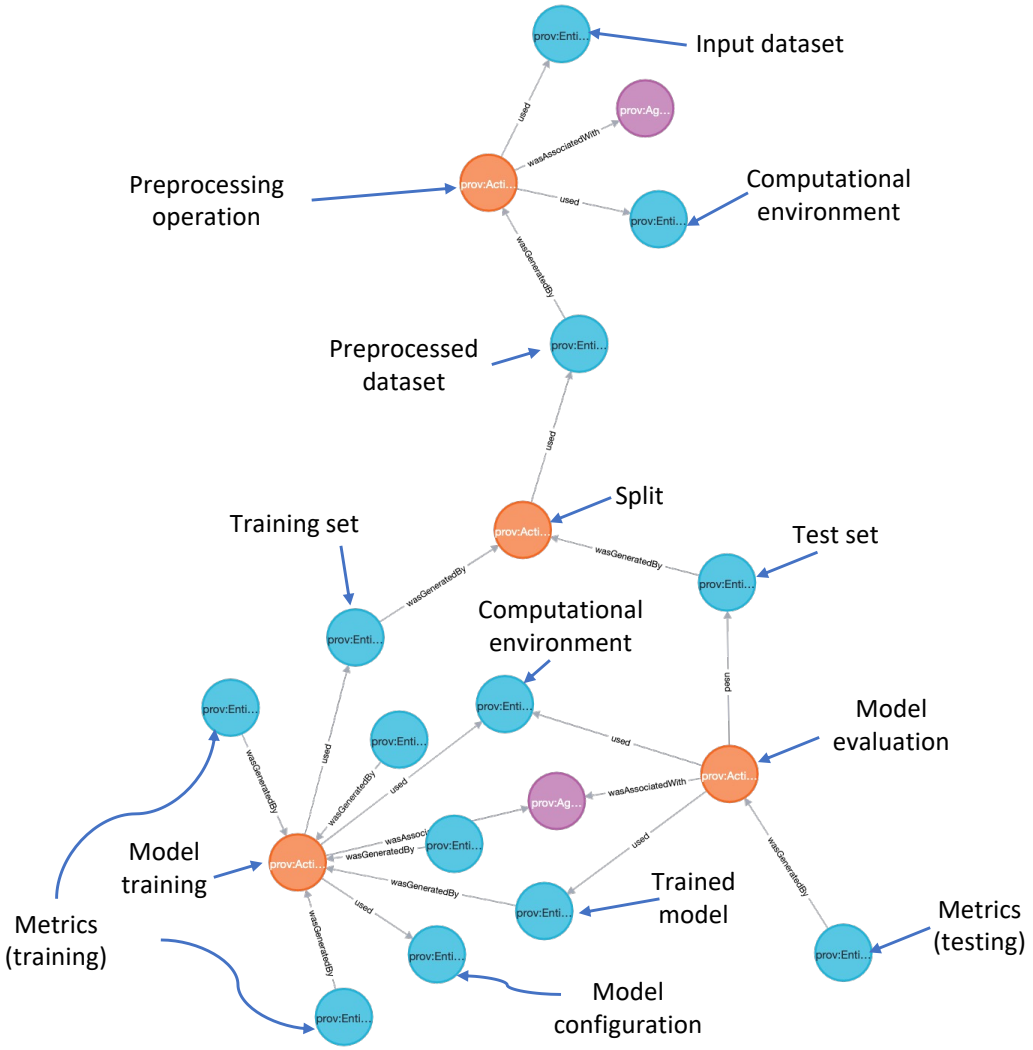


Figure 5.5: DLProv provenance in Neo4j graph for a single AlexNet model training after preprocessing the data.

a cross mark (✗) symbol indicates that the approach is not able to answer the query, either because the relevant data is not captured or the necessary relationships between elements are not established. The symbol ± denotes that the approach partially answers the query. The colors in Table 5.3 distinguish between entity-level and relationship-level queries. Light gray represents entity-level queries, which can be answered using metadata or by retrieving the input/output of a single DL workflow activity. Dark gray indicates relationship-level queries, which require tracing the lineage of artifacts. These are characteristic provenance queries, often involving graph traversal to explore dependencies between entities and activities within the workflow. MLflow allows analyses during training through its user interface and using SQL when a database is available in the backend; MLflow2PROV allows the generated provenance graph to be stored for further processing in specific provenance storage systems or graph databases (*e.g.*, Neo4J which can be queried with the graph query language Cypher). DLProv allows queries to be submitted to MonetDB while training the model, and to Neo4j after the model is trained. The data

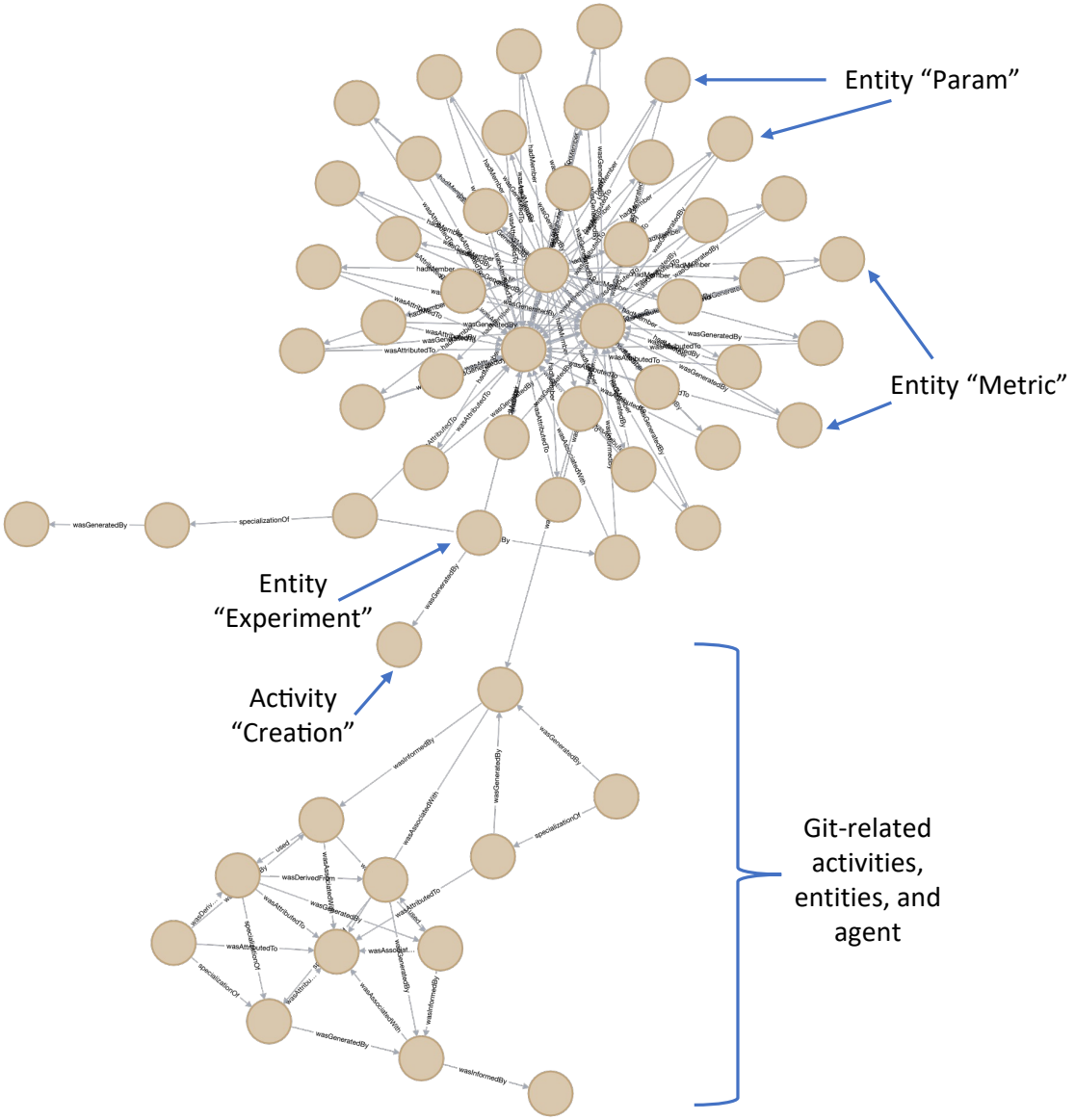


Figure 5.6: MLflow2PROV provenance in Neo4j graph for a single AlexNet model training after preprocessing the data.

captured by DLProv and stored in its *DL Model Provenance* can also be analyzed using visualization libraries such as Kibana.

DLProv can answer Q1, using both SQL and Cypher, by filtering all trained models to which the entity *iTraining* has `learning_rate = 0.002`. According to the instantiated example in Figure 5.4, this result would be the entity *iTesting_101* that has a relationship of *wasGeneratedBy* with the activity *Training_101*, that *used* learning rate as 0.002 in the *iTraining_101* in Figure 5.4.

To obtain the average of a specific metric of a workflow run in the MLflow UI, direct calculation within the MLflow UI is not possible. In such cases, scientists must perform this processing using external tools or scripts, typically by exporting the data. Consequently,

Table 5.3: Query support of DLProv, MLflow, and MLflow2PROV.

Query	MLflow	MLflow2PROV	DLProv
Q1	✓	✓	✓
Q2	✓	✗	✓
Q3	✓	✗	✓
Q4	✓	✓	✓
Q5	✗	✗	✓
Q6	±	±	✓
Q7	✓	✓	✓
Q8	✓	✓	✓
Q9	✓	✓	✓
Q10	±	✗	✓
Q11	✗	✗	✓
Q12	✗	✗	✓

queries that require calculating the metric average Q2 and Q3 are not feasible within the MLflow UI, unless the average itself is logged as a metric. However, these queries can be performed using SQL, querying the tables *metrics* and *params*. Q3, for instance, can be answered by submitting a query to retrieve the average accuracy, then selecting the highest one, and joining this result with the table that contains the hyperparameters of this specific run. On the other hand, MLflow2PROV only generated provenance with the first and last training epoch. Therefore, we are unable to find the average processing time or the maximum training accuracy, which would answer Q2 and Q3, respectively. DLProv can answer Q2 using both SQL and Cypher. Using Cypher, it can be simplified as `MATCH (a:Entity) RETURN avg(a.‘dlprov:elapsed_time’)` when we are considering just a single model. The result of this query (in this example, the time in seconds) can be represented as the following JSON: `[“avg(a.‘dlprov:elapsed_time’)”: 10.414]`. DLProv follows the same idea as MLflow to answer Q3, which can be performed in both SQL and Cypher.

The scores for the evaluation step (*i.e.* test set) are not captured using the auto-logging API of MLflow. Therefore, we were only able to answer query Q4 after adding annotations to capture this metric. Regarding Q4, a query that requires the relationship between different steps of the DL workflow, although minimum or maximum values cannot be filtered in the MLflow UI, this query can be easily answered using SQL. However, in a scenario where there is not a considerable number of trained models in an experiment, it is possible to make that analysis through the charts provided by MLflow. After identifying the model with the highest test accuracy, the hyperparameters can be checked in the “Overview” tab. Since with the annotation, MLflow captures and stores this metric, MLflow2PROV generates its provenance also with this information of type *Metric*. Therefore, the scientist, after finding the highest test metric, can relate it to all the entities

of type *Param* to find the hyperparameters. DLProv can easily answer this query using SQL and the aggregation function *max()* and then relate the result to the hyperparameters table. Using Cypher, it results in a derivation path from the *oTesting_101* (let us say that this is the entity with attribute test accuracy with the highest value), that *wasGeneratedBy Testing_101* that *used* the trained model *iTesting_101* that *wasGeneratedBy* the activity *Training_101* that *used* the set of hyperparameter *iTraining_101*.

We are also unable to answer Q5 with MLflow2PROV and MLflow, since we could not gather the data that was used to train the model. In the MLflow UI as well as SQL, we can find some information about the data, such as feature shape and size. DLProv allows Q5 to be submitted to MonetDB, getting the highest test accuracy using `max()`, and relating to the data used to train or evaluate that model, as stored by the entities generated by the activity *SplitData_101*. This query can also be submitted to Neo4j.

Following a similar approach as Q2, we can answer Q6 using MLflow by executing an SQL query to retrieve the minimum loss value using the *min()* function and then joining it with the `params` table to obtain the corresponding hyperparameter values. As in Q5, some information about the dataset used can be retrieved, but not at a detailed level. With MLflow2PROV, this query can be answered by identifying the *Metric* entity with the minimum loss and tracing its relationships to the corresponding *Param* entities to extract the associated hyperparameters, similar to Q4. DLProv enables answering this query through both SQL and Cypher queries by following the derivation trace from the *oTesting_101* entity to the *iTraining_101* and *iDataSplit_101* entities, providing a structured way to trace the dataset, hyperparameters, and evaluation metrics associated with the best-performing model.

Q7 can also be answered through the MLflow UI by checking the hyperparameter in the "Overview" tab of a run or using SQL to select the parameters of a specific run like `SELECT * FROM params WHERE run_uuid=:value`. MLflow2PROV can answer this query following the same derivation path as Q1. To answer Q7, DLProv considers the entity related to the hyperparameters that were *used* in the activity training (*Training_101*). The graph that represents the outcome of this query follows the entity *iTesting_101* that *wasGeneratedBy* an activity *Training_101* that *used* an entity *iTraining_101* containing the hyperparameters. DLProv can also answer this query using SQL.

MLflow captures information about the environment where the training was performed, such as Python and Keras versions, as necessary to answer query Q8. This information can be found in the "Artifacts" tab. However, information regarding the hardware is not currently available. MLflow2PROV represents this information in an entity of type *RunTag* and name *mlflow.log-model.history*. To gather this information for a specific run, a scientist can submit a query like `MATCH (n:Entity{'prov:type':`

`'RunTag', name: 'mlflow.log-model.history'}) RETURN n`, or use the *shortestPath* function to find the path from that to the entity *RegisteredModel*. To answer Q8, DLProv follows the same rationale as Q7, but instead of the hyperparameters used in the DL model training, we are interested in the computational environment *used* in the activity *Training_101*, named *dlprov:170.229.25.114*. This entity contains information about the software used to preprocess the data, to train the model (like MLflow), or even about the hardware, such as whether CPUs or GPUs were used in a specific run.

Q9 is related to the scientists (*i.e.* agents) involved in the training steps of the ML workflow. Although MLflow does not directly capture the metadata of the scientist, the tool does contain the login of the machine where the training was performed, answering Q9. MLflow2PROV uses these data captured by MLflow by defining an agent in the form `agent(User?name\=&email\=, [name="", email="", username ="", username="debora", prov:type= "User"])` and associates with the activity of model creation. In DLProv, this query follows the same rationale as the previous ones; however, in this case, we are interested in the Agent that *wasAssociatedWith* the training activity. This query can be written as `MATCH (a:Agent)<-[:wasAssociatedWith]-(b:Activity{'meta:identifier_original':'Training_101'}) RETURN a, b`.

For Q10, preprocessing operations applied to the input data are considered. Data preprocessing is a step (*i.e.*, an activity) in the DL workflow. Therefore, instead of saving preprocessing operations as parameters of a run in MLflow, we logged this code as an artifact. However, even with this information, Q10 is partially answered since we can look for the hyperparameters in the MLflow using both UI and SQL. To find the preprocessing Python code saved as an artifact, we would have to go through the local directory, and there is no definition of a relationship between the input data, the preprocessing, and the model training. Therefore, Q10 cannot be fully answered. We could not find this piece of information in the MLflow2PROV graph in Neo4j. In DLProv, Q10 can be tackled in SQL, during model training, obtaining dataset metadata and hyperparameters. After training, a query using Cypher can provide the preprocessing operations applied to the data.

MLflow does not capture how the data was split into train, validation, and test. Therefore, Q11 and its modified version to gather the data used to evaluate the trained model cannot be answered. Since MLflow2PROV generates its provenance based on MLflow, these queries cannot be answered using MLflow2PROV as well. DLProv follows the data derivation path from the trained model *iTesting_101* that *wasGeneratedBy* the activity *Training_101* that *used* a set of data, in this case, images, identified by *oTrainingTuples_-101*, to answer Q11. The modified Q11 results in an entity by following the relationship

used from the activity *Testing_101* to the entity *oTestingTuples_101*. For these queries, the level of detail in the answer can vary depending on the granularity of provenance captured in the preprocessing step. A coarse-grained answer might contain the dataset name and source, while a fine-grained answer could provide the images used to train and evaluate the trained model.

Q12 follows the same pattern as Q10 when considering MLflow and MLflow2PROV. Consequently, neither MLflow nor MLflow2PROV can answer this query. Although a recent work (SCHLEGEL and SATTLER, 2024) states that MLflow2PROV can extract data preprocessing and feature transformation flows from pipeline runs by leveraging attributes in the Transform Activity, we were unable to answer this query using MLflow2PROV at the time of these experiments. Q12 concerns which preprocessing operations were applied to the input data (represented in the *iInputDataset_101* entity). Since DLProv allows for the integration of these activities, it can be answered using Cypher. The result of this query presents a data derivation path from the entity with the trained model (*iTesting_101*) to the activities that *iInputDataset_101 used* in the preprocessing, which in this example is just one (*Operation1_101*).

From Table 5.3, we observe that MLflow and MLflow2PROV face challenges in answering relationship-level queries, which require tracing dependencies between entities and activities. While both can handle entity-level queries by retrieving metadata and direct attributes of DL workflow components, they lack traceability capabilities. Although MLflow2PROV cannot address all queries involving relationships between different workflow steps, it marks a valuable effort toward enhancing analysis capabilities in DL. Its structured representation of experiments helps understand the different executions. MLflow2PROV focuses primarily on capturing provenance related to versioning, rather than tracking the training process in detail, functioning more as a post-execution tool than as an integrated provenance solution.

Some of these queries retain their significance even after the DL model has been deployed and require ongoing maintenance (*e.g.*, dealing with data drift). In such scenarios, DLProv generates the PROV-N document or even the Neo4j database autonomously, irrespective of the framework that produced the DL model. This encapsulates the concept of the model's provenance, not found in the solutions analyzed. Without this provenance document, scientists need to revert to the solution that captured the data, like MLflow, retrieve data related to the deployed model, and run the query there, a process that indicates a lack of autonomy in provenance analysis.

5.3 Traceability in Different Scenarios

The experiments in this section focus on showing DLProv's independence from specific DL frameworks and computational environments, addressing RQ3. They highlight DLProv's ability to capture provenance across different steps of the DL workflow, including data preparation, DL model training, and evaluation, using DL frameworks such as TensorFlow, PyTorch, and DeepXDE, while maintaining a consistent provenance representation. This section also shows how DLProv can be used in diverse computational environments, including Google Colab, Lobo Carneiro, Santos Dumont, Grid5000, and personal laptops, as presented earlier.

5.3.1 Integrating DLProv for Keras and Google Colab

This experiment focuses on showing the feasibility of using DLProv within Google Colab, enabling GPU-based execution and supporting the analysis of provenance data in DL workflows. We evaluate the benefits of DLProv through experiments with AlexNet and DenseNet, which were executed in the Google Colab cloud. These experiments use matplotlib to generate charts from the provenance data stored in the DLProv database, showcasing DLProv's flexibility in integrating with visualization libraries.

Google Colab is a tool that allows the user to integrate Python source code with text (usually in markdown). This type of environment is commonly called a "Notebook". The advantage of notebooks is that they provide a collaborative environment with zero configuration effort and access to several types of resources. Although there are some disadvantages of using Colab for some ML experiments (*e.g.*, a series of automatic settings are not suitable for ML experiments), the integration of DLProv for Keras and Colab seems to be promising, especially because Colab provides access to several generations of GPUs. This type of hardware is not easily available in commodity machines.

To integrate DLProv for Keras with Google Colab, an external environment had to be set. Although it is possible (and easier) to deploy both the provenance system and the ML application on the same machine, this is not recommended by Google Colab. The reason is that when using Colab, the execution depends on the user's active session (which has a timeout), and data persistence may last for more time than the set timeout. In addition, it is not recommended to install third-party components that depend on a Database Management System (DBMS), *e.g.*, MonetDB, and to execute Java Web Application (*e.g.*, query interface). A cloud solution was chosen to host both DLProv for Keras (*i.e.*, the provenance system) and MonetDB. DigitalOcean[9] was elected because it offers resources with low financial cost and simplicity, providing easy installation of containers. MonetDB and

[9] `https://www.digitalocean.com/`

DLProv for Keras, which are deployed as a single container, were installed in a private virtual machine, and their services were configured to be accessible from Google Colab through a public IP address.

Figure 5.7 presents the architecture that integrates DLProv for Keras with Google Colab and DigitalOcean (*i.e.*, external environment). Although there is a communication overhead between both virtual machines (in Colab and DigitalOcean), this configuration has the benefit of separating ML experimentation and provenance infrastructure. Inside an enterprise, for example, there are different expertise and responsibilities: the production team would support the provenance system hosted in an appropriate infrastructure, on-premise or cloud-based, while the data science team would be end-users, training new applications and submitting provenance queries using Python notebooks transparently.

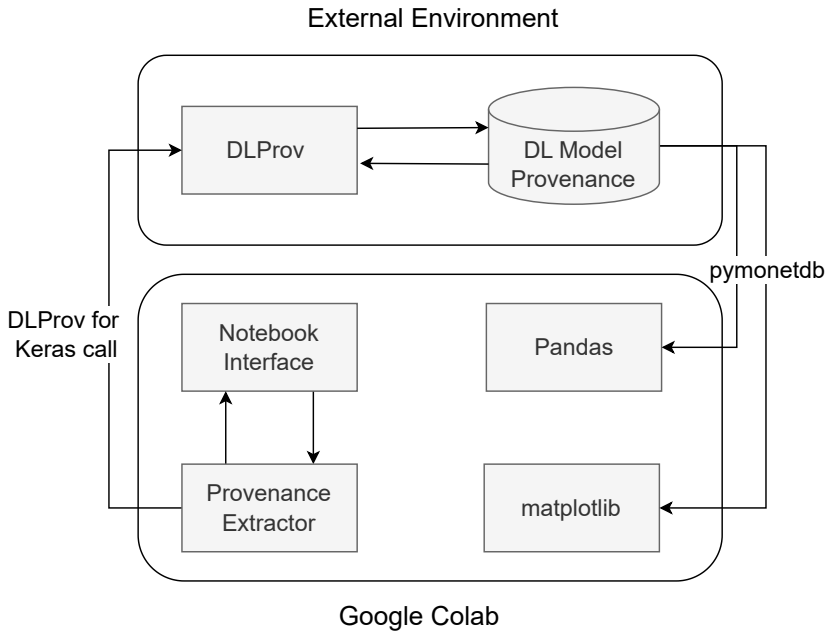


Figure 5.7: Architecture that integrates Google Colab with DLProv for Keras libraries and services.

We evaluate the integration of DLProv for Keras and Google Colab with two real DL experiments. The first experiment uses AlexNet to evaluate aspects of data monitoring and queries when using Google Colab and to analyze the limitations of deploying the MonetDB DBMS, as the DLProv for Keras provenance persistent system, to the Colab notebook. The second experiment used DenseNet to explore data analytical issues.

Differently from AlexNet, presented in Section 5.2.2, Dense Convolutional Networks (*i.e.*, DenseNet) (HUANG *et al.*, 2017) connect each layer of the neural network to every other layer in a feed-forward way. This way, DenseNet presents L(L+1)/2 connections, which is different from the L connections found in most neural networks. The use of

DenseNets in many problems presents several advantages, *e.g.*, they foster feature propagation and (possibly the most important advantage) reduce the number of parameters. In our experiments, a pre-trained DenseNet model with the ImageNet dataset is used as a base layer with fixed parameters. On top of this pre-trained model, we add a trainable dense layer with dropout, batch normalization, and max-pooling, as well as a last dense layer with softmax activation.

Although the integration of DLProv for Keras and Colab is promising and opens room for many refinements, it presents a drawback regarding resource reservation. Since Colab provides free computational resources, it automatically adjusts the hardware availability at runtime (*e.g.*, during the training process of a neural network). This is a behavior already found in many cloud providers, *e.g.*, the AWS Spot market (PORTELLA *et al.*, 2019). Although Colab offers ways to provide on-demand virtual machines that guarantee performance, this type of resource was not used in these experiments. Colab also offers many types of GPUs to use, however, the types of GPUs that are available vary over time. By the time the experiments were executed, the GPU configuration provided by Colab was based on a Tesla T4 (which implements the Turing architecture). In addition, it is not guaranteed that the experiments will be executed on a dedicated card. The amount of memory in Colab virtual machines also varies from 12GB to 25GB over time, but the amount of memory does not vary during the life cycle of a specific virtual machine.

Similar to most DL experiments, we need to vary a set of hyperparameters to get the best results possible. For AlexNet, we have chosen to evaluate the impact of the activation function choice. Seven different activation functions are explored, *i.e.*, Rectified Linear Activation (ReLU) (NAIR and HINTON, 2010), Logistic (Sigmoid), Softmax (GOODFELLOW *et al.*, 2016), Softplus (GLOROT *et al.*, 2011), Softsign (GOODFELLOW *et al.*, 2016), Hyperbolic Tangent (Tanh) and Scaled exponential linear unit (SELU) (KLAMBAUER *et al.*, 2017). The hyperparameter settings for AlexNet are presented in Table 5.4. The type of optimizer, the learning rate, the number of epochs, and the dropout value are fixed, and the activation function of the internal layers of the AlexNet network is varied. Thus, experiments are performed for each of the aforementioned activation functions. All these hyperparameters are automatically captured by DLProv for Keras.

Table 5.4: AlexNet Hyperparameter Configuration.

Fixed Parameters				**Varied Parameters**
Optimizer	**Learning Rate**	**Epochs**	**Dropout Rate**	**Activation Function**
Adam	0.0001	100	0.4	ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU

With DenseNet, two experiments are performed. In the first, different activation functions are evaluated in the dense layer of the network, as presented in Table 5.5. While in AlexNet the activation functions are related to multiple layers, in this experiment, the

activation function that is varied is the one at the dense layer (except for the final dense layer) used on top of the pre-trained layers. In the second experiment, different learning rates and dropout rates are evaluated, while the number of epochs, the optimizer, and the activation function are fixed, as presented in Table 5.6.

Table 5.5: DenseNet Hyperparameter Configuration - Experiment 1.

Fixed Parameters				**Varied Parameters**
Optimizer	**Learning Rate**	**Epochs**	**Dropout Rate**	**Activation Function**
Adam	0.0001	100	0.4	ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU

Table 5.6: DenseNet Hyperparameter Configuration - Experiment 2.

Fixed Parameters			**Varied Parameters**	
Optimizer	**Activation Function**	**Epochs**	**Dropout Rate**	**Learning Rate**
Adam	Softplus	10	0.2, 0.4, 0.6, 0.8	0.001, 0.01, 0.1

It is worth noticing that the volume of data used in the experiments is relatively small, and no cross-validation techniques were used. Thus, many of the analyses on the accuracy of the models may be the result of random oscillations and, therefore, should not be seen as effective conclusions about the overall performance of the models. The experiments presented are data collected, including all executions in which the environment had to be set up. We used the free version of Colab, the computational environment was shared among many users, altering computational performance and measurements. Also, using two environments generated the additional cost of sending external calls to DLProv. It is noteworthy that, despite this, there is no harm to the goals of this experiment, which are to explore the use of the DLProv for Keras integrated with Google Colab and not to evaluate the effectiveness of the models.

As aforementioned, all collected provenance data is stored in the Provenance Database in MonetDB. In the Colab notebook, `pymonetdb` is used to retrieve the provenance data stored by DLProv for Keras. A series of queries can be submitted using `pymonetdb`. In this experiment, a SQL query is submitted to retrieve the accuracy and the loss for each evaluated activation function. Based on this provenance query result, Figure 5.8 presents the accuracy and loss charts generated with `Matplotlib`. By analyzing the results in Figure 5.8, one can state that the activation function that obtained the best accuracy and lowest loss is the Softplus activation function. This network configuration presents a maximum accuracy of 76% and a minimum loss of 1.19, and both values occurred at epoch 50. The performance of the accuracy and loss of other functions over the epochs can be seen in Figure 5.8.

To further explore the capabilities and limitations of DLProv for Keras, two experiments were carried out with DenseNet. The DenseNet model was pre-trained with the ImageNet

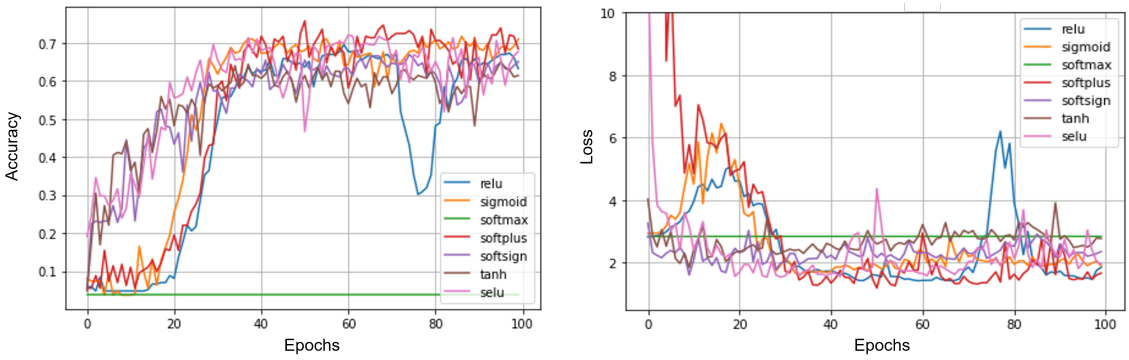


Figure 5.8: Accuracy and Loss over the epochs for the different activation functions used in AlexNet.

dataset, and we use this pre-trained model as a base layer with fixed parameters, while on top of this model, we add a trainable dense layer. In the first experiment with DenseNet, we analyze which activation function presents the best accuracy and loss. While in the AlexNet experiment, the activation functions are related to multiple layers of AlexNet, in this experiment, the activation function that is changed is the one at the only trainable dense layer used on top of the pre-trained DenseNet. Table 5.7 shows the two epochs that present the best accuracy and loss, in both of them, Softplus is the activation function. Thus, the Softplus activation function is the choice for the second experiment with DenseNet.

Table 5.7: The best accuracy among all explored activation functions for DenseNet.

Model	Activation Function	Learning Rate	Dropout Rate	Epoch	Validation Accuracy	Validation Loss	Processing Time
DenseNet	Softplus	0.01	0.4	17	0.863	0.529	13.77
DenseNet	Softplus	0.01	0.4	18	0.863	0.527	13.79

Having the same provenance representation (*i.e.* database schema) helped us to compare results from AlexNet and DenseNet. Also, the fact that DLProv for Keras follows a public recommendation facilitates integration with applications, allowing users to analyze and manage data in a way that suits their needs.. DLProv for Keras associates performance execution data to the provenance data, like the execution time of each (or set of) epoch and the total training time. Table 5.8 presents the average processing time (*i.e.*, $\overline{x}$) in seconds with AlexNet and DenseNet for each network configuration. DenseNet presents a more complex architecture when compared to AlexNet, so it requires higher processing times. In the experiments, both networks consider learning rate 0.0001, dropout 0.4, 100 epochs, and the following activation functions: ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh, SELU.

Finally, in the second experiment with DenseNet, the activation function (*i.e.*, Softplus), the Optimizer (*i.e.*, Adam), and the number of epochs (*i.e.*, 10) are fixed. We varied the

Table 5.8: Comparison of the average processing time of epochs of each optimizer for different networks.

Activation	Alexnet+DLProv for Keras	DenseNet+DLProv for Keras
ReLU	3.107	13.833
Sigmoid	3.121	13.839
Softmax	3.772	13.848
Softplus	3.217	13.834
Softsign	3.207	13.850
Tanh	3.204	13.863
SELU	3.839	13.351
$\overline{x}$	**3.352**	**13.77**

dropout rate (0.2, 0.4, 0.6, and 0.8) and the learning rate (0.001, 0.001, and 0.1) to evaluate the accuracy and the loss for each combination. Figure 5.9 and Figure 5.10 present the accuracy and loss for each combination, respectively. These charts were generated using data captured by DLProv. This type of analysis is fundamental for the user to set the proper number of epochs for a specific experiment. In many cases, adding more epochs does not help much in improving the accuracy of the model.

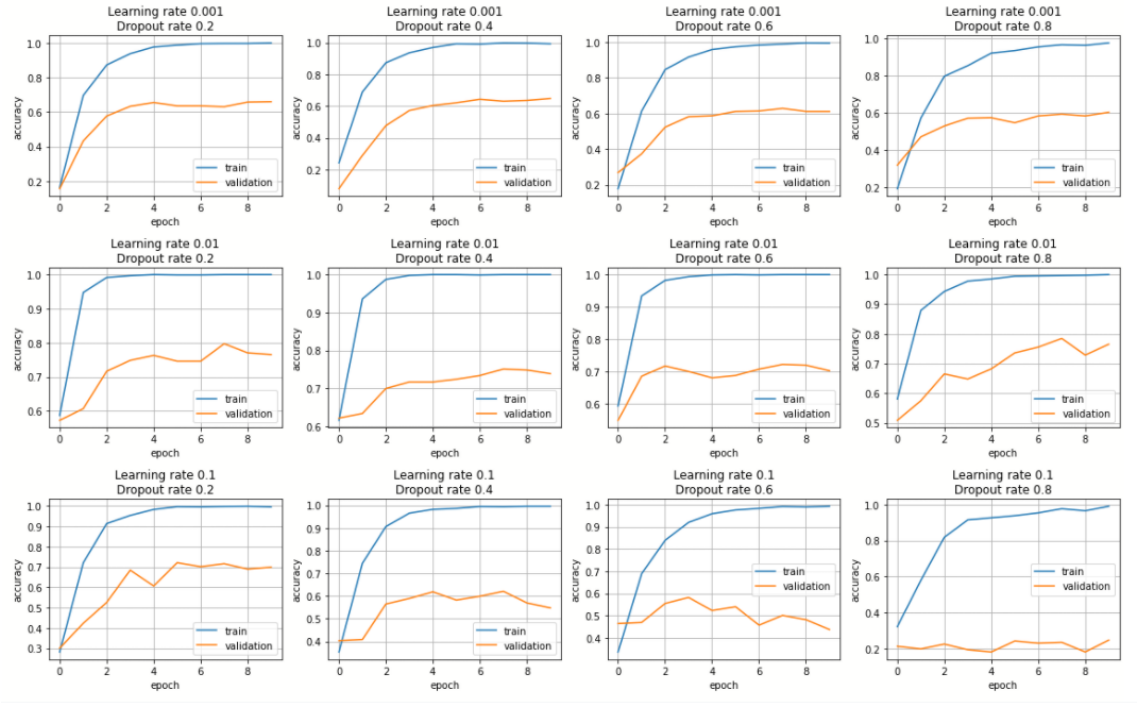


Figure 5.9: Accuracy rates obtained for training and validation of each combination of parameters with DenseNet.

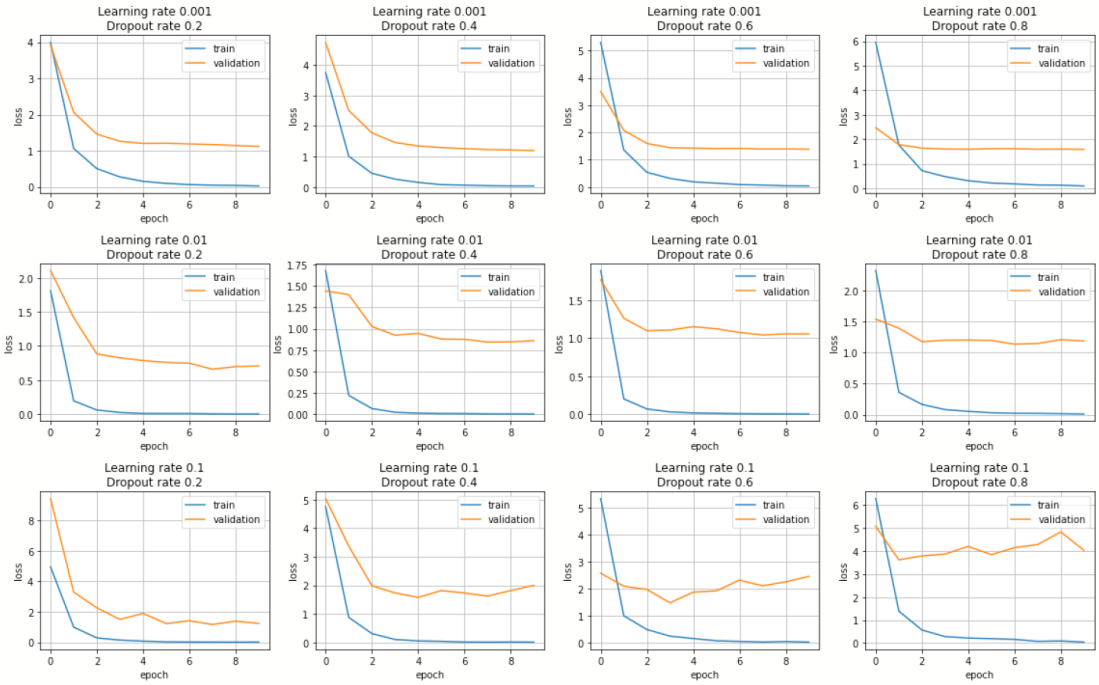


Figure 5.10: Loss rates obtained for training and validation of each combination of parameters in Experiment 2 with DenseNet.

5.3.2 Simple Deep Learning Model on the MNIST Dataset

This experiment focuses on showing the use of DLProv in a PyTorch-based DL workflow, evaluating the overhead introduced by provenance capture, and addressing its ability to answer queries that can be performed during DL model training. It involves training a simple DL model on the MNIST dataset and shows DLProv's compatibility with Parsl[10], a script parallelization tool used to distribute tasks across two GPUs. This showcases DLProv's ability to support DL workflows running on HPC systems and parallelized with different libraries.

In this experiment, we implemented a code for training a DL architecture using PyTorch. The DL architecture was trained on the MNIST dataset[11]. The source code for these experiments was divided into functions and executed using Parsl via the *@python_app* decorator. These functions were organized as follows:

- **Data Loading:** The dataset was loaded and preprocessed using `load_data()` task.
- **Model Building:** DL architecture was defined in the `build_model()` task.

[10] `https://parsl.readthedocs.io/en/stable/index.html`
[11] `https://pytorch.org/vision/0.20/generated/torchvision.datasets.MNIST.html`

- **Model Training:** The model was trained in the `train_model()` task on both GPUs over 50 and 100 epochs, using the Adam optimizer and a learning rate of 0.001.
- **Model Evaluation:** The model was evaluated in `evaluate_model()` task on the test set to measure performance.

The DL training was performed using several hyperparameter configurations. These configurations included different learning rates and optimizers, each applied to preprocessed data with distinct sequences. Performance metrics such as accuracy, loss, and training time were captured throughout the training process. Such performance metrics can be analyzed during training through the execution of queries in MonetDB to answer questions such as, "What is the hyperparameter configuration used to train the model with the highest test accuracy?". By analyzing the results of different configurations, insights can be gained regarding the impact of hyperparameters on model performance.

A quantitative experiment was conducted to provide a comparative analysis of the cost of capturing provenance data using DLProv and MLflow during the training of a DL model. Specifically, the execution time of the workflow, from loading data to evaluating the trained model, was analyzed under three conditions: *(i)* without capturing provenance data (baseline), *(ii)* with provenance data capture, including relationships, using DLProv, and *(iii)* with metadata capture using MLflow. The experiments were conducted on an Ubuntu 22.04.4 LTS machine, with an Intel (R) Core (TM) i9-14900KF processor, featuring 32 logical CPUs and 2 NVIDIA GeForce RTX 4090 GPUs, each with 24 GB. We identify this machine as *Ubuntu Machine*.

Overview

The MNIST dataset consists of a training set with 60,000 examples of handwritten digits, covering the 10-digit classes, and a test set with 10,000 examples. For this experiment, we used a simple DL model. The architecture consists of an input layer designed to handle 2D grayscale images with a shape of 28x28 pixels, as found in the MNIST dataset. This is followed by a *Flatten* layer, a *Dense* layer, a *Dropout* layer, and a final *Dense* layer. We chose to train a simple DL architecture on the MNIST dataset because it is a well-known benchmark in the field of ML. The dataset offers an accessible starting point for those interested in exploring learning techniques and pattern recognition methods with real-world data while requiring minimal effort in terms of preprocessing. Given that our goal is to show the analysis of DL workflows during model training, we found that MNIST provided an ideal context for showcasing these analyses.

Overhead Analysis

Figure 5.11 shows the execution times in seconds for the simple DL model with MNIST. To obtain a reliable average, each experiment was executed 10 times. In the 50-epoch experiment with 2 GPUs, DLProv introduces minimal overhead compared to the Baseline, with an overhead of approximately 1.392%, which is quite small. MLflow introduces an overhead of about 5.841%. These results indicate that DLProv and MLflow have a relatively low impact on the execution time when compared to the Baseline. The standard deviation for each solution (3.37 for Baseline, 1.96 for MLflow, and 1.95 for DLProv) shows that the times for DLProv and MLflow are fairly consistent, with a slightly higher variance for the Baseline. For the 100-epoch experiment with 2 GPUs, DLProv again shows a minimal overhead of about 1.393%, and MLflow's overhead is approximately 4.489%. The standard deviations for this round (4.03 for Baseline, 3.63 for MLflow, and 4.30 for DLProv) reflect a slightly higher variation in execution times compared to the 50-epoch experiment.

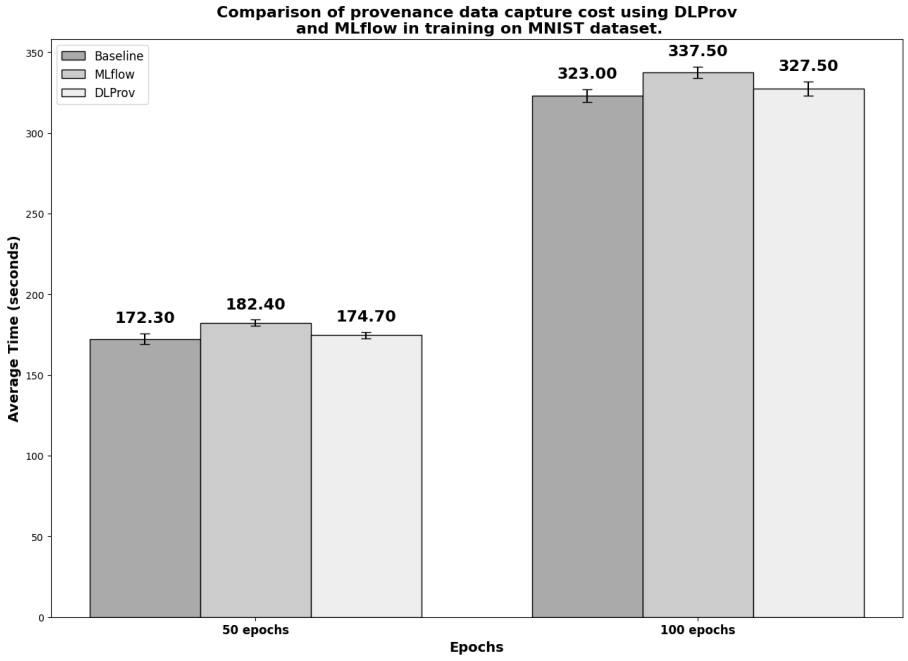


Figure 5.11: Execution time for 50 and 100 epochs - SimpleNN with MNIST.

Provenance Query Analyses

While training a simple DL model with the MNIST dataset, a data scientist's goal is to identify the best candidate models and, ultimately, select the most suitable model for deployment. This process involves exploring hyperparameter configurations and applying different preprocessing techniques. To make informed decisions, it is important to analyze and compare the performance and characteristics of the trained models. DLProv facilitates this analysis by enabling queries on its DL Model Provenance database in

MonetDB. To support these analyses, we submit queries Q1, Q2, Q3, Q4, and Q5, which provide insights into hyperparameters, performance metrics, and the data contributing to the models. Additionally, other queries from Table 4.1 can be leveraged during this stage to deepen the analysis.

Q1 focuses on identifying all trained models that used a specific learning rate, such as 0.001. As shown in Figure 4.3, this can be achieved by joining the *itrain* and *otrained-model* classes. The result of this query in SQL provides details about the trained models, including an identifier, the model name, and the file path where the model is saved (*e.g.,* /home/dbpina/parsl-exps/parsl-experiments/Resnet50/resnet50-trained.keras). This approach allows us to track and compare models based on their learning rate configuration.

Q2 can also leverage the join between the *itrain* and *otrainedmodel* tables. First, we filter the specific DL model by criteria such as learning rate (*e.g.*, 0.001) and batch size (*e.g.*, 32), or apply additional filters as needed. Then, by joining with the *otrain* table, we can calculate the average epoch processing time by aggregating the elapsed time for the corresponding training task, using the task identification associated with the DL model. This approach enables us to track and analyze the training performance, providing insights into the efficiency of the model training process.

Q3 follows a similar approach to Q2, but in this case, the goal is to identify the DL model with the highest training accuracy. To achieve this, we first calculate the average training accuracy for each model by averaging the accuracy values in the *otrain* table. We then find the maximum of these average accuracies. Using this maximum value, we can identify the specific DL model that achieved the highest training accuracy. This is done by performing a join between the *itrain* and *otrainedmodel* tables based on the *train_task_id*, allowing us to retrieve the corresponding hyperparameters and other relevant details for the model with the highest training accuracy. Listing 10 shows a snippet of this query.

Q4 also uses the *MAX* function in SQL, but unlike Q3, there is no need to compute an average since there is only one test accuracy value for each model. In this case, the goal is to identify the DL model with the highest test accuracy. By directly applying the *MAX* function to the test accuracy values in the *otest* table, we can find the model with the highest performance on the test set. We then perform a join with the *itrain* table, based on the *train_task_id*, to retrieve the corresponding hyperparameters and other details of the model that achieved the highest test accuracy.

Q6 delves further back into the derivation trace, focusing on the data that contributed to the highest test accuracy. Similar to Q4, which identifies the model with the highest test accuracy, Q5 examines the underlying data used in the training and evaluation of that

```
SELECT im.*, om.avg_accuracy
FROM itrain im
JOIN (
    SELECT train_task_id, AVG(accuracy) AS avg_accuracy
    FROM otrain
    GROUP BY train_task_id
    HAVING AVG(accuracy) = (
        SELECT MAX(avg_accuracy)
        FROM (
            SELECT AVG(accuracy) AS avg_accuracy
            FROM otrain
            GROUP BY train_task_id
        ) AS subquery
    )
) om ON im.train_task_id = om.train_task_id;
```

Listing 10: Example of SQL syntax for querying Q3, illustrating the structure and components of a query designed to retrieve specific information about hyperparameters and accuracy.

model. The query begins by identifying the highest test accuracy stored in the *otest* table. Once this value is determined, a join with the *otrainedmodel* table is performed to locate the specific DL model that achieved this accuracy, based on *test_task_id*. Finally, the query links this model to the *odata_split* table, using *train_task_id*, which holds information about the datasets used during training and evaluation. This process enables us to trace back to the exact data that contributed to the training and testing of the model with the highest test accuracy, providing a comprehensive view of the data's role in achieving this result.

5.3.3 ResNet50 Architecture on the CIFAR-100 Dataset

This experiment focuses on using DLProv in a TensorFlow-based DL workflow, measuring the overhead introduced by provenance capture and showing how it supports provenance queries. This experiment, also performed on the *Ubuntu Machine*, involved training the ResNet50 architecture on the CIFAR-100 dataset[12]. The dataset was accessed through Keras' built-in small datasets module[13]. Similar to the first experiment, this one followed the same process of organizing the source code into functions, which were applied via the *@python_app* decorator with Parsl, covering key tasks such as Data Loading, Model Building, Model Training, and Model Evaluation. In addition, a quantitative evaluation

[12] https://www.cs.toronto.edu/~kriz/cifar.html
[13] https://keras.io/api/datasets/cifar100/

was performed to conduct a comparative analysis of the cost of capturing provenance data using DLProv and MLflow, mirroring the approach taken in the first experiment.

Overview

The CIFAR-100 dataset has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses, such as aquatic mammals, fish, and flowers. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

We chose ResNet50 (HE *et al.*, 2016) for our second experiment, a deep residual network, due to its widespread adoption in the research community. By using ResNet50, we aimed to assess the provenance capture provided by DLProv in a more complex, state-of-the-art DL model in comparison to the simpler feed-forward neural network used in the first experiment. We used a model that builds on the ResNet-50 architecture available on the Keras webpage[14]. The ResNet50 model is used as a feature extractor, with the top classification layer removed ('include_top=False').

Overhead Analysis

Figure 5.12 shows the execution times in seconds for the ResNet50 with CIFAR-100. To obtain a reliable average, each experiment was executed 10 times. The 50-epoch experiment with 2 GPUs shows that DLProv adds minimal time compared to the Baseline (without provenance capture), with an overhead of about 1.002%, and MLflow adds 4.265% of overhead. In the 100-epoch experiment with 2 GPUs, DLProv's overhead is even lower, at approximately 0.519%, and MLflow showed an overhead of about 5.672%.

During the initial runs of the 50-epoch experiment, the Baseline unexpectedly took longer than DLProv. This led us to investigate CPU usage, which showed fluctuations. In addition, many CPU cores were idle, causing inconsistent performance. To address this, we set CPU affinity to "block" in Parsl to assign adjacent cores to workers, and limited the experiments to use only 15 cores, which were the most active ones during training. After this change, the execution times became more consistent, and the Baseline ran as expected. Despite this, the Baseline's runtime still exhibited a relatively high standard deviation for 50 epochs at 13.35 seconds, slightly higher than MLflow's 13.13 seconds, and significantly higher than DLProv's more stable runtime with a standard deviation of 7.13 seconds. For 100 epochs, the standard deviation increased across all cases: the Baseline showed 21.94 seconds, MLflow 33.12 seconds, and DLProv 25.45 seconds. Both DLProv and MLflow introduce overheads that can be considered negligible compared to the

[14] `https://keras.io/api/applications/resnet/#resnet50-function`

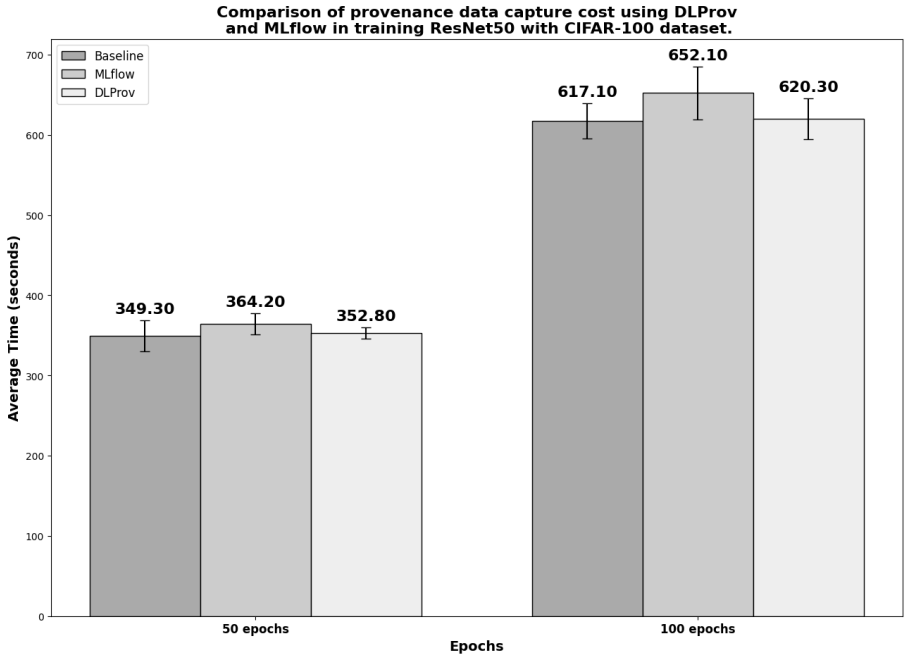


Figure 5.12: Execution time for 50 and 100 epochs - ResNet50 with CIFAR100.

DL workflow elapsed time and the benefits of provenance capture. However, it is worth noting that MLflow stores each parameter and metric in separate files, which may explain the slightly higher overhead due to the increased number of I/O operations.

Provenance Query Analyses

For the ResNet50 training with CIFAR-100, consider a scenario where we trained different model configurations and obtained model results. We have analyzed and selected a DL model among the candidates to be deployed. Following the completion of the experiment, with the selection of the DL model to be deployed, a provenance document was generated with the steps that led to it. This graph includes information such as the hyperparameters used, the preprocessing steps applied, and details of the DL model architecture. By capturing the full provenance, the graph is a traceable record of the model development process, enhancing transparency in a production environment.

The provenance document, available in PROV-N or JSON, is generated from the provenance data stored in MonetDB immediately after the DL model is selected for deployment. Once generated, the document is packaged with the DL model. Queries can then be submitted by importing the provenance document into Neo4j, leveraging the functionality provided by DLProv, or using a dedicated solution such as the PROV Database Connector[15]. This setup allows for querying the provenance graph. Figure 5.13 shows the provenance graph for the deployed DL model in Neo4j. The blue circles represent entities, the orange circles denote activities, and the purple circles correspond to agents.

[15] `https://github.com/DLR-SC/prov-db-connector/tree/master`

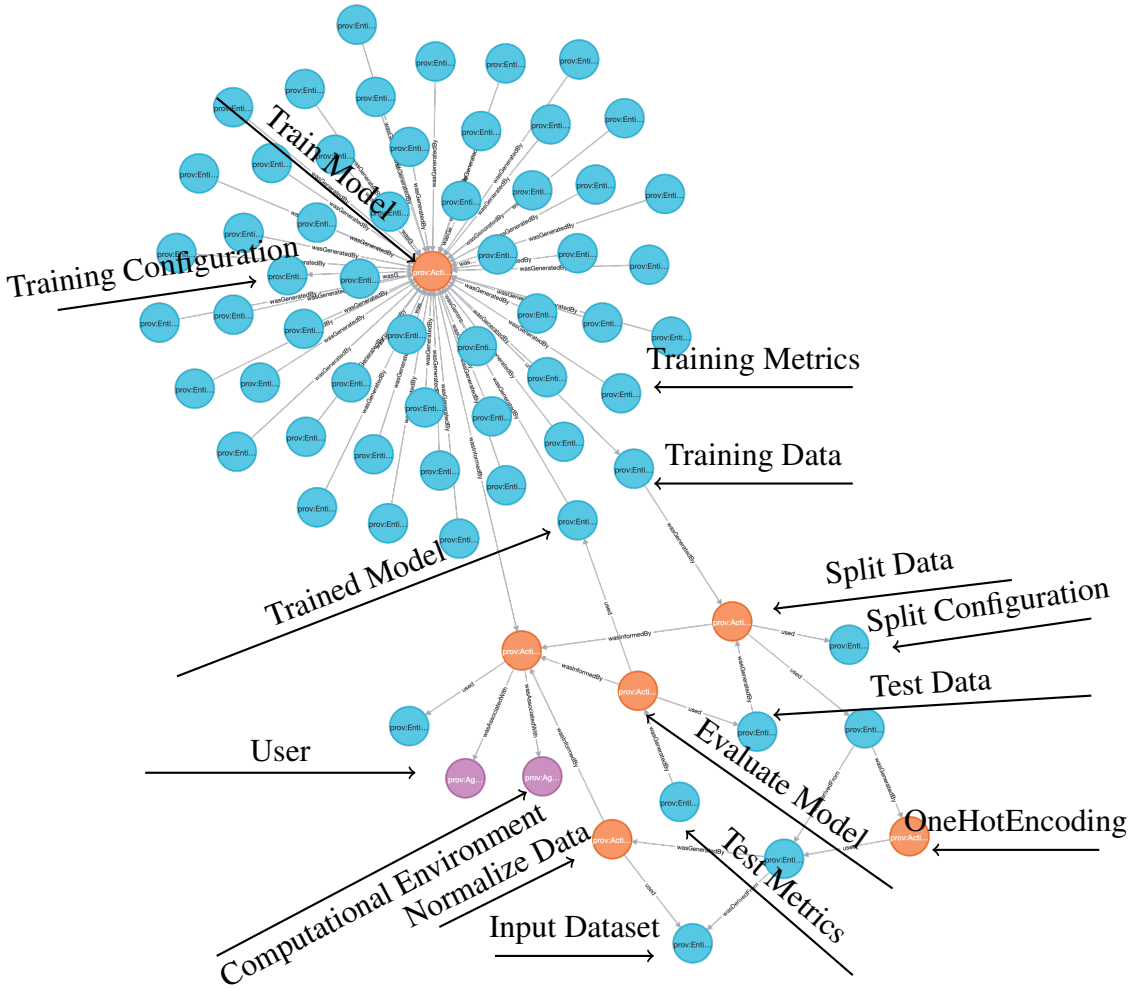


Figure 5.13: Provenance graph in Neo4j showing the deployed DL model's derivation trace.

For this deployed DL model, we submit queries Q7, Q8, Q9, Q10, Q11, and Q12. It is important to note that some of these queries can also be performed during the generation of DL models and training. In such cases, they can be answered using MonetDB.

Query Q7 seeks to retrieve the hyperparameters used in the deployed DL model, identified as *M4*. During the provenance document's creation, DLProv assigns a unique identifier to each model; in this case, it is *dlprov:99ad4e82-17b1-49f1-86ad-b52cd7dcab6e*. However, we will avoid using this identifier for better readability. The response to this query involves the entity *oTrainedModel*, which captures the details of M4, the DL model resulting from training. This entity is linked to the activity *Train* through the *wasGeneratedBy* relationship and is associated with the hyperparameters entity *iTrain* via the *used* relationship. The hyperparameters recorded include the optimizer Adam, learning rate of 0.001, 100 epochs, 179 layers, and a batch size of 32.

Query Q8 focuses on identifying the computational environment used to train a DL model. Similar to the approach for Q7, this query retrieves information from the provenance data

associated with the activity *Train*. However, instead of targeting the hyperparameters, the emphasis is on the computational environment *used* during the training of model M4. The response reveals that the computational setup included the operating system Ubuntu 22.04.4 LTS, an i9-14900KF processor, 15 CPUs, and 2 NVIDIA GeForce RTX 4090 GPUs. Even though details about the computational environment used in experiments are typically known in advance, representing this information in the provenance graph is important, especially when comparing how the same DL model behaves under different computational setups.

Query Q9 seeks to identify the Agent that *wasAssociatedWith* the activity *Train*. Similar to the approaches for queries Q7 and Q8, this query focuses specifically on the agent responsible for the training process. The response involves the entity *oTrainedModel*, which is connected to the activity *Train* via the *wasGeneratedBy* relationship. This activity, in turn, is linked to the agent *User* through the *wasAssociatedWith* relationship. The agent's information includes the name and may also include additional details, such as an email address, providing accountability and traceability for the model training process.

Query Q10 investigates the entire process that led to the creation of model M4, tracing all activities and entities involved in producing this deployed DL model. The answer to this query relies on navigating the provenance graph to identify the sequence of activities, their associated agents, and the entities they *used* or *generated*. Starting from the entity *oTest*, which contains the set of metrics that *wasGeneratedBy* the activity *Test*, which *used* the test set to evaluate the *oTrainedModel*, which represents model M4. We traverse its *wasGeneratedBy* relationship to the activity *Train* that *used* a training set. From here, we can explore all connected inputs and outputs, such as the preprocessed datasets, hyperparameters, computational environment, and intermediate steps, such as the activity *SplitData*. Additionally, it captures the feature transformations applied to the preprocessed dataset and links back to the original input dataset. This traversal provides a complete derivation trace, detailing how each element contributed to the model's creation.

Query Q11 retrieves the dataset used to train the deployed DL model. Using the DLProv provenance graph, we can trace the derivation path starting from the trained model, represented by the entity *oTrainedModel*, which *wasGeneratedBy* the activity *Train*. This activity *used* a specific dataset, identified as *oTrainSet*. In this case, the dataset comprises images stored in the file located at the path *temp_cifar100/train.npz*.

Query Q12 focuses on identifying the preprocessing operations applied to the input data, represented by the entity *iInputDataset*. With the integration capabilities provided by the DLProv suite, which connects different steps in the DL workflow, this query can be executed using Cypher. The result offers a derivation trace starting from the trained model entity *oTrainedModel*, traversing back to the preprocessing activities associated with *iIn-*

putDataset. For instance, in the training of ResNet50 on the CIFAR-100 dataset, two preprocessing operations were applied. The resulting preprocessed dataset *oOneHotDataset* *wasDerivedFrom* a *OneHotEncoding* activity, which, in turn, *used* an entity *oNormalizeData* that *wasDerivedFrom* a *NormalizeData* activity. The latter directly *used* the original input data *iInputDataset*.

Although the provenance capture for these experiments was conducted using different DL frameworks, PyTorch and TensorFlow, it is important to emphasize that the provenance data representation remains consistent. This consistency facilitates interoperability, enabling analysis and comparison of provenance information across different DL frameworks.

Some of the queries discussed, whether with SQL or Cypher, cannot be answered using frameworks, such as MLflow and MLflow2PROV, that fail to capture relationships that allow this type of analysis, and even the level of detail. This limitation arises because MLflow focuses primarily on high-level experiment tracking, such as logging parameters, metrics, and artifacts, without integrating detailed provenance information about the data transformations, computational environments, or derivation traces of entities. Similarly, MLflow2PROV extends MLflow by mapping its tracking information to PROV, but it relies on the scope of MLflow's original data. Queries like Q5, which require tracing data contributions through a detailed derivation trace, or Q10, which investigates the derivation process of a DL model, require a richer provenance graph that captures all steps and entities in the DL workflow. These queries highlight the need for more comprehensive provenance solutions, such as DLProv, which explicitly integrates fine-grained data and workflow provenance into its model.

5.3.4 Handwritten Transcription Workflow Execution

This experiment focused on the inference step for transcribing handwritten Portuguese texts[16]. Unlike the previous workflows, which were centered around model training, this experiment involved inference, where a trained model is used to make predictions. To accommodate this shift, the DLProv suite's provenance model was extended to capture the activities and entities specific to the inference workflow. In this workflow, we added a comparison step, where the inferred text is compared to the ground truth, which was included primarily to generate performance metrics for provenance analysis. Although metrics are typically associated with training and evaluation rather than inference, incorporating them here allows us to show how DLProv can capture and query relationships between inputs, outputs, and performance in an inference scenario. This experiment was conducted on the *Ubuntu Machine*.

[16] `https://github.com/MeLLL-UFF/handwriting-transcription-ptbr/tree/master`

Overview

In this case study, the handwritten transcription process involves several steps: reading an input image, detecting regions within the image that contain textual information, recognizing the text within each region, and transcribing it into a file. These transcriptions are then combined to reconstruct the full text. To evaluate the processes described, we used the Brazilian Forensic Letter Database (FREITAS *et al.*, 2008), consisting of 945 image samples of the same text, written by 315 authors, with each author having written three texts by hand. The pages containing the text are scanned in grayscale, generating the images contained in the database. These images undergo an initial processing to obtain only the words contained in each text. To do this, a technique is applied for detecting word areas based on the identification of characters in images and their respective affinity regions, combined to form the region that corresponds to a word. From the demarcation of these spaces, an additional process extracts the words, resulting in a new set of images, each containing one word. In this set, certain regions were detected that encompass more than one word, which results in an image containing both. Because of this, a second process is performed to separate these words into distinct images. In other cases, due to aspects inherent to the writing style of the text, certain regions of words are not identified, and a technique is proposed to change aspects of the image in an attempt to make the information contained therein more expressive, approaching common characteristics of images in which text identification is possible. The final set of word images is provided as input to perform the task of recognizing textual information and return as output the recognized words and their respective probabilities of being the words originally contained in the images. To validate the recognized words, we compared the inferred text to the original text and computed the accuracy of this inference. Figure 5.14 shows the activities in this workflow.

Provenance Query Analyses

Figure 5.15 shows a fragment of the provenance graph generated from the inference of a single sample. For clarity, some nodes have been omitted. This provenance graph traces de handwritten transcription workflow, starting with loading the dataset (*LoadData*) and a pre-trained model (*LoadPretrainedModel*), followed by crafting word regions (*RunCraft*), cropping images (*CropWordsFromImage*), and performing inference (*RunInference*). The last activity in the provenance graph is a comparison (*RunComparison*) between the inferred and the original text, capturing metrics such as accuracy. The activities *RunCraft* and *CropWordsFromImage* are part of the data preparation step in the DL workflow. In this experiment, we show not only the capability to extend the data to be captured but also DLProv's ability to capture preprocessing operations while seamlessly integrating with DL model training or inference processes.

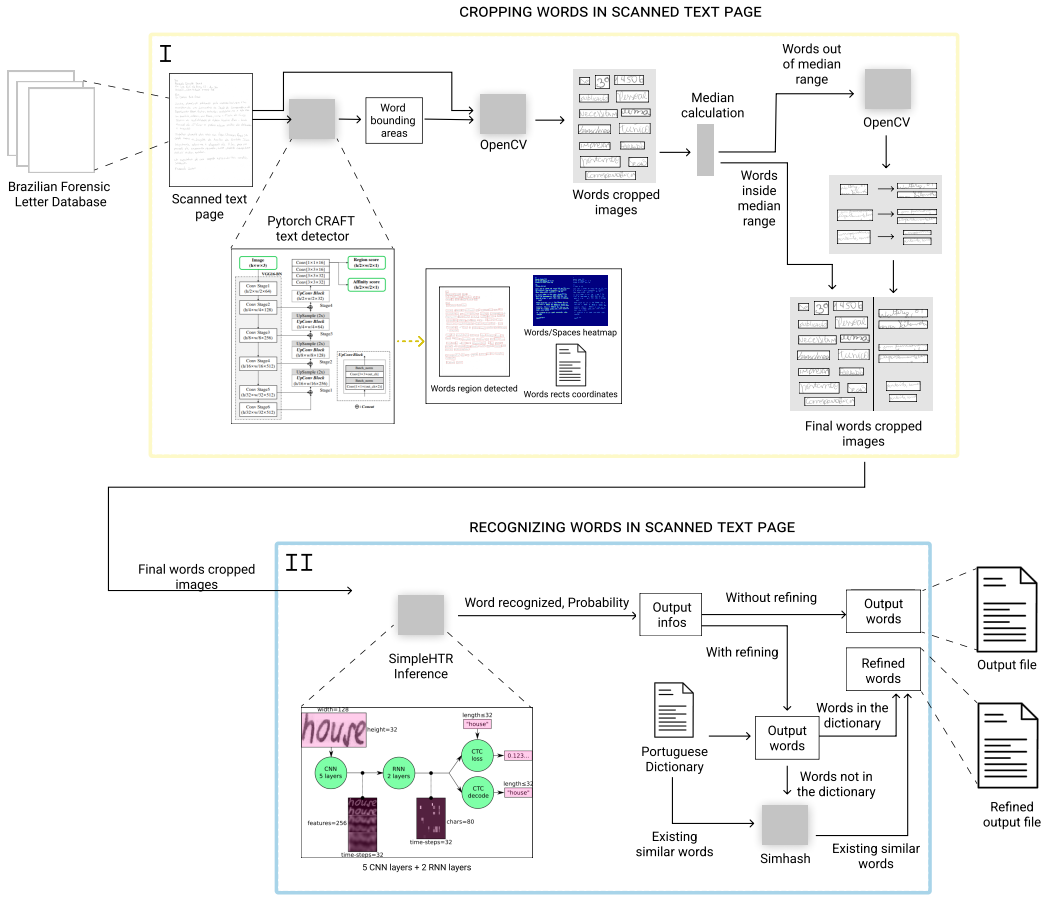


Figure 5.14: Workflow for transcribing handwritten Portuguese (DA SILVA *et al.*, 2023a).

To evaluate the provenance captured by DLProv in this experiment, we can adapt the provenance queries from Table 4.1 to fit this context. For instance, while Q6 is typically used during DL model development, it can be modified for the inference step to analyze relevant metrics, identified as *inference_metric* in entity labeled as *"dlprov:ds_tag":"oruncomparison"*, which serves as a key evaluation measure here. Since we captured accuracy instead of loss, we can adapt the query to *Given a set of samples of the same text, what are the sample, threshold, and model decoder values and the evaluation measure values associated with the inference that has the highest accuracy?*. This query is performed over multiple executions, not only on the one presented in Figure 5.15. To answer this, we first need to find the highest accuracy (*inference_metric*) and then trace the entity containing this highest value, which *wasGeneratedBy* the activity *RunComparison*. From there, we trace back to the entities *oRunCraftOutput* containing the sample, *iModelInput* containing the decoder, and *iRunCraftInput* containing the threshold, which together led to this accuracy.

This query can also be adapted to answer *Given sample Sample_CF00001_01, what are the values of the threshold used and the accuracy achieved?*, maintaining the same struc-

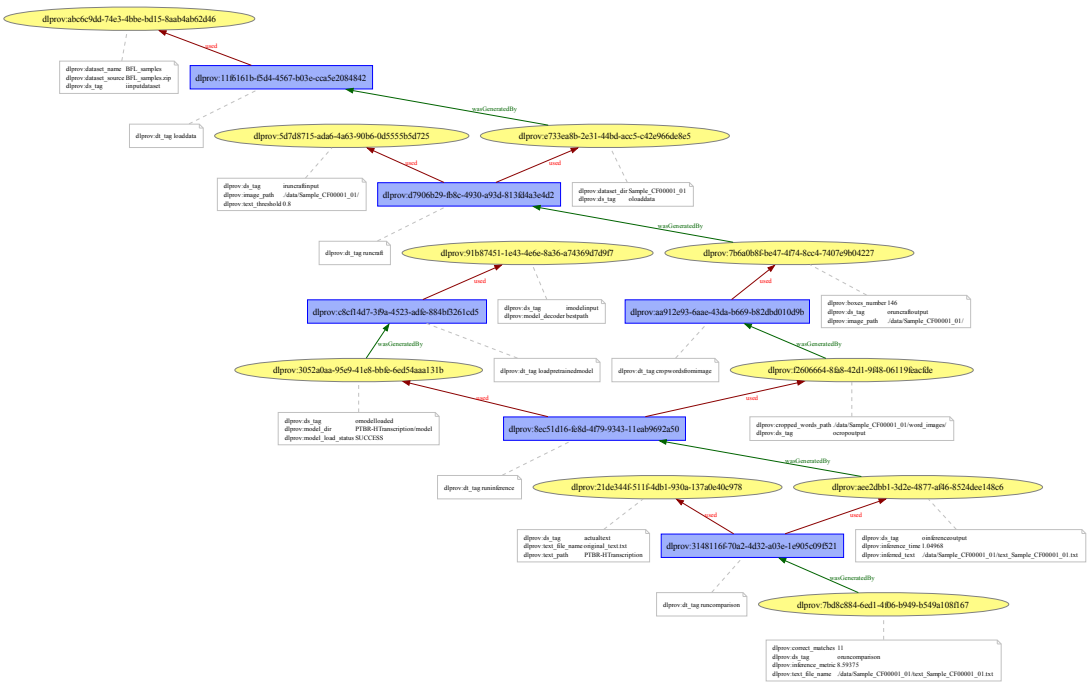


Figure 5.15: Fragment of the W3C PROV provenance graph showing the derivation trace of the inference of a sample in the workflow for transcribing handwritten Portuguese.

ture as the original query by relating the input data (the sample), the parameter (the threshold), and the metric (the accuracy). The answer to this query follows a similar derivation process to the previous graph analysis. It reveals that a threshold of 0.4 resulted in an accuracy of 6.25%, while a threshold of 0.8 achieved an accuracy of 8.59%.

In this experiment, without a provenance represented as a graph document that can be queried, there is no information about the trained model being used for inference. We do not know the type of data it was trained on, the hyperparameter values used, or how the data was prepared. In the query example presented, we used different threshold values, which led to a variation in the accuracy. This raises the question: could this difference be caused by a mismatch between the inference process and the original training process, of which we have no knowledge? This lack of transparency highlights the need for a provenance document to accompany the DL model, as shown in the ResNet50 experiment. Such documentation would provide traceability into the training process, including details on data, hyperparameters, and data preparation, allowing for a better understanding of discrepancies during inference.

5.3.5 DenseED

The focus of this experiment is to present the use of DLProv in a domain-specific application involving scientific DL, highlighting its applicability beyond standard DL tasks. This

case study uses the neural network DenseED and its datasets, as proposed by (FREITAS *et al.*, 2021). DenseED uses a CNN as a surrogate model to enable the quantification of uncertainties (ZHU and ZABARAS, 2018). The network architecture is a dense CNN as shown in Figure 5.16. DenseED aims to replace the calculation of the Reverse Time Migration (RTM) equations with a trained model. In this way, the RTM calculation that would have to be performed for each probability distribution can be reduced. DenseED's architecture adopts a fully convolutional Bayesian encoder-decoder network. The number of dense layers in DenseED is variable, and its evaluation metric is the Mean Squared Error (MSE). This case study requires more data to be analyzed than the DLProv default hyperparameter values. Therefore, additional modeling and instrumentation were required to track the model's architecture. The provenance graph representation of the dense blocks and their transformations can be seen in Figure 5.17.

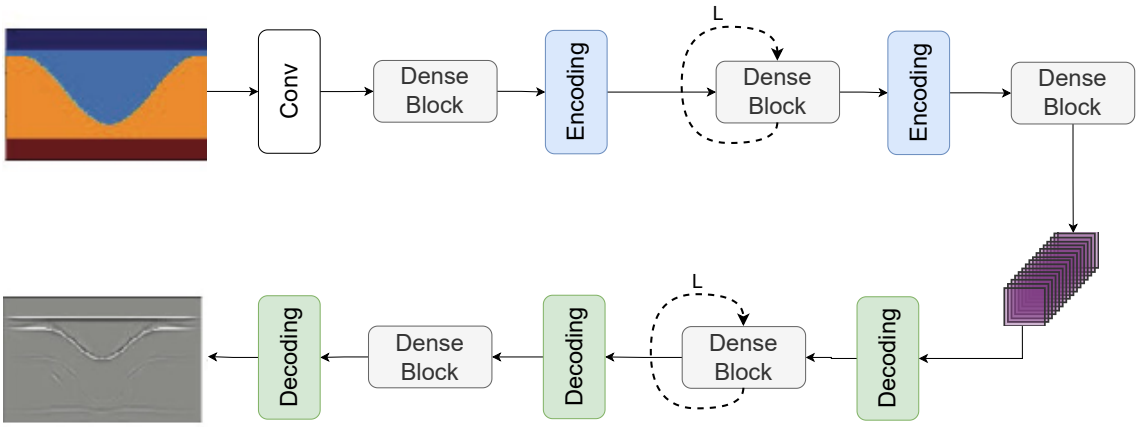


Figure 5.16: DenseED: Deep Convolutional Encoder-Decoder network architecture from (FREITAS *et al.*, 2020)

These experiments were performed on the Lobo Carneiro computer cluster, also known as LoboC, from the High-Performance Computing Center (NACAD) at COPPE/UFRJ. LoboC is an SGI ICE-X Linux cluster with 504 Intel Xeon E5-2670v3 (Haswell) CPUs, totaling 6,048 processors. The processors feature Hyper-Threading (HT) technology, offering 48 processing threads per node, with 64GB of RAM. Compute nodes are interconnected with InfiniBand FDR–56 Gbs (Hypercube) technology. The cluster runs under a shared disk architecture, with an Intel Luster parallel file system with 500TB storage capacity. The DL Model Provenance database was deployed on a dedicated computational node for managing provenance data, while DenseED used four nodes at LoboC.

As reported in (FREITAS *et al.*, 2021), the user conducted several hyperparameter tunings. Without provenance, it is time-consuming to gather the different training executions to compare and register adjustments. In this case study, we reexecuted some of these trainings with the provenance support from DLProv to show its analytical support. The input dataset of DenseED contains 100,000 velocity fields. From the input dataset, 12000 sam-

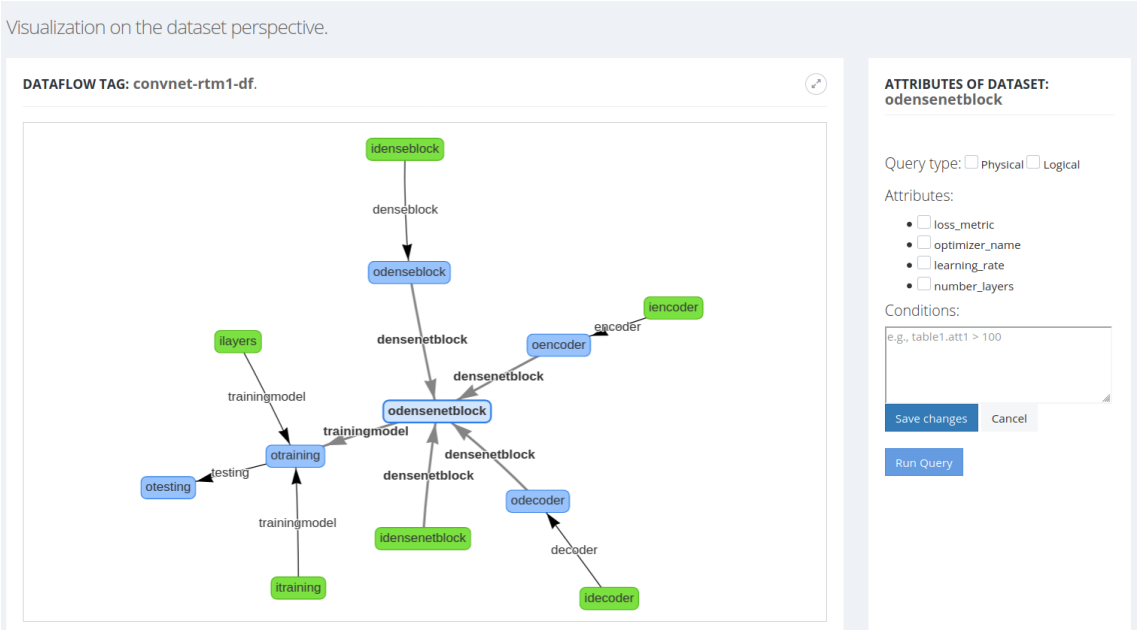


Figure 5.17: Provenance Viewer example with DenseED Dataflow.

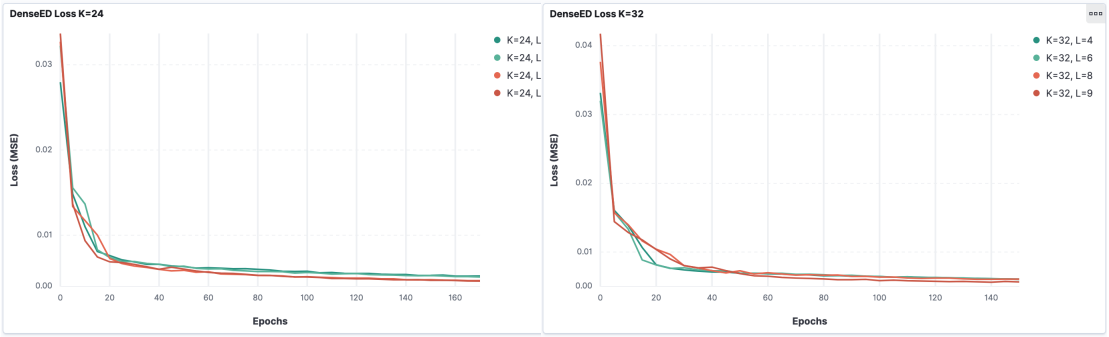
ples were randomly selected for the training and 1000 for the testing set. In the validation of this NN, 50% of the training set was used. This neural network generates a surrogate model for RTM, thus, the expected output is a seismic image. The loss in this model is the Mean Squared Error (MSE), and it is monitored during the training and the testing phases, where the Coefficient of Determination (R^2) is the main metric in the test phase. The training explores alternatives for the Growth Rate ($k = (16, 24, 32)$) and the number of layers in the dense block ($l = (4, 6, 8, 9)$). All this data is stored in the provenance database.

In Figures 5.18a, 5.19a and 5.19b, we show the DenseED training results with different k and l values. In the cases explored, after finishing the training of combinations where $k = 16$, the user observes in Figure 5.18a that at epoch 140 and beyond, there is a tendency that MSE values stay within a range of 0 and 0.001, zooming the chart (Figure 5.18b) and querying the database, this tendency is confirmed. Assuming the same behavior for the combinations of $k = 24$, he decides to decrease the number of epochs from 200 to 170 epochs. As expected, in Figure 5.19a, the MSE from epoch 140 stabilizes within values 0 and 0.001, and then the user trains the $k = 32$ combinations with 150 epochs. These changes in the number of epochs to be trained decrease the resource consumption and the execution time of combinations of $k = 24$ and $k = 32$, with no significant change in MSE. Looking at these results through queries is also possible, but the amount of information presented in a table would take longer to generate insight and a decision.

The user can submit analytical queries to tune DenseED parameters like “What are the initial learning rate, optimizer, and the number of layers when the training for DenseED

(a) (b)

Figure 5.18: (a) Graphical view of the training results of DenseED with $k = 16$ and $l = (4, 6, 8, 9)$ (b) Zoomed view of the training results for DenseED with $k = 16$ and $l = (4, 6, 8, 9)$ from epoch 140 to epoch 200.



(a) (b)

Figure 5.19: (a) Graphical view of the training results of DenseED with $k = 24$ and $l = (4, 6, 8, 9)$ (b) Graphical view of the training results of DenseED with $k = 32$ and $l = (4, 6, 8, 9)$.

achieved the highest R^2?", "Retrieve the combinations of k and l that achieved the top three MSE values and their R^2.", "What are the top five training MSE values and their elapsed time for each epoch when $k = 32$ and $l = 9$?", "What are the combinations of k and l that consume less time during the training?". Tables 5.9,5.10, 5.11, and 5.12 show query results. Table 5.9 shows the characteristics of the model that had the best value for R^2, and Table 5.10 might suggest that greater values of l lead to smaller MSE values. The results of Table 5.11 might be a case where, if more values were required, a graphical representation would be more helpful. Table 5.12 shows that the training time increases with the number of layers (l). This is important for the user to evaluate the tradeoff between time cost and the value of R^2. For DenseED, the user considers that $R^2 \geq 0.95$ as a satisfactory R^2. The combination $k = 16$ and $l = 4$ was chosen since it already reached the user's criteria and costs less time. Queries that filter and order results to show the highest and lowest values assist in evaluating extreme values and outliers as the training evolves. Without this data, keeping track of this progress associated with execution data and the hyperparameter set would be costly and error-prone.

Table 5.9: What are the initial learning rate, optimizer, and the number of layers when the training for the combination of k and l that achieved the highest R^2?

Initial Learning Rate	**Optimizer Name**	**Number of Layers**	k	l	R^2
0.01	Adam	118	32	9	0.9979

Table 5.10: Retrieve the combinations of k and l that achieved the top three MSE values and their R^2

MSE	R^2	k	l
0.00093	0.9979	32	9
0.00103	0.9976	16	9
0.00111	0.9975	24	8

Table 5.11: What are the top five training MSE values and their elapsed time for each epoch when $k = 32$ and $l = 9$?

Epoch	**Elapsed time**	**MSE**
144	41.7	0.000589
138	41.93	0.000609
142	41.71	0.000612
141	41.74	0.000617
145	41.8	0.000619

Table 5.12: What are the combinations of k and l that consume less time during the training?

Average elapsed time	k	l
7.3	**16**	**4**
9.2	**24**	**4**
11	**32**	**4**
13.2	16	6
17.1	24	6
20.3	16	8
21.5	32	6
25	16	9
28.1	24	8
34.1	24	9
35.6	32	8
43.8	32	9

5.3.6 PINN Eikonal

This experiment aims to show how DLProv can be used in a different domain-specific DL setup, such as scientific DL workflows. The experiments were performed on both Grid5000 (BALOUEK *et al.*, 2013) and the Santos Dumont supercomputer (SDumont). Grid5000 is a large-scale, flexible testbed designed for experiment-driven research, with a focus on parallel and distributed computing, including Cloud, HPC, Big Data, and AI. The experiments on Grid5000 used a CPU-GPU hybrid computing mode. SDumont, at the time of the experiments, had approximately 5.1 Petaflop/s of processing capacity,

using a hybrid configuration of computational nodes that incorporate different parallel processing architectures. SDumont has a Lustre parallel file system, with a raw storage capacity of about 1.7 PB. In the experiments, we used one partition named GDL, which is specialized for artificial intelligence tasks. GDL has two processors, Intel Xeon Skylake Gold 6148 2.4GHz with 20 cores, eight GPUs NVIDIA Tesla V100-16GB with NVLink, and 384GB of RAM.

DLProv for PINNs was used with a PINN that solves the Factored Eikonal Equation (FEE) SILVA *et al.* (2021b), implemented in TensorFlow, which is a first-order nonlinear equation relevant in many fields. It describes phenomena like wave propagation for acoustic and elastic media, as well as electromagnetic media DEBNATH (2012). Therefore, the Eikonal equation plays an important role in problems like optics, shortest path problems, image segmentation, and seismic and medical imaging. The Eikonal equation is given by,

$$\|\boldsymbol{\nabla}\phi(\mathbf{x})\|_2 = \frac{1}{v(\mathbf{x})}, \forall \mathbf{x} \in \Omega. \tag{5.1}$$

where $\Omega \subset \mathbb{R}^{n_{sd}}$, is the domain, $n_{sd} = 1, 2, 3$ is the number of space dimensions, $\mathbf{x} = \{x, y, z\}$ is the position vector, and $\|.\|$ stands for the L^2-norm. Figure 5.20 shows the two neural networks that were used, one to estimate the transit time (TT), related to the direct problem, and the other to estimate the wave propagation velocity (Vel), related to the inverse problem. These networks are indirectly connected to calculate the loss function.

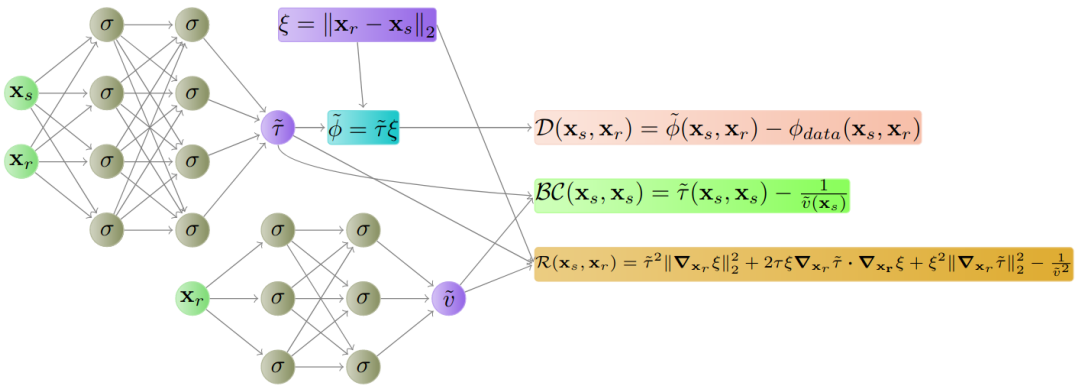


Figure 5.20: PINN Eikonal scheme, where $\phi(\mathbf{x}_s, \mathbf{x}_r)$ represents the transit times, and $v(\mathbf{x}_r)$, the propagation speed of the wave in the acoustic medium. They are approximated by two different neural networks, their approximations are denoted by $\tilde{\phi}(\mathbf{x}_s, \mathbf{x}_r)$ and $\tilde{v}(\mathbf{x}_r)$ respectively. Those approximations are then fed to the loss components related to the data assimilation, boundary, and initial conditions, and the PDE residual SILVA *et al.* (2021b).

The input dataset refers to seismic and ground-penetrating radar data. The metric is R^2, which is known as the coefficient of determination. The weights w of the loss function are

defined as in Equation 5.2. The loss function $\mathcal{L}$ has three components related to $\mathcal{L}_{\mathcal{D}}$ the data assimilation, $\mathcal{L}_{\mathcal{BC}}$ as boundary and initial conditions, and $\mathcal{L}_{\mathcal{R}}$ as the PDE residual.

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \mathcal{L}_{\mathcal{R}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{R}}) + \mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{BC}}) + 25 \cdot \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{D}}) \tag{5.2}$$

Provenance Query Analyses

In the first experiment, conducted on Grid5000, several training runs were performed with the FEE PINN, with variations in the activation function (ReLU, Tanh, Sine, Sigmoid) and optimizer (Adam, RMSProp). We used provenance to monitor metrics by epoch during the PINN training and steer the training at runtime, inspecting how the evaluation metrics are evolving, including the loss, $\mathcal{L}$, and its components, $\mathcal{L}_{\mathcal{BC}}$, $\mathcal{L}_{\mathcal{R}}$, and $\mathcal{L}_{\mathcal{D}}$. Thus, we can change parameters and start training again if, for instance, the loss value is not meeting a chosen criterion.

During the online data analysis for the current training (optimizer Adam and activation function Tanh), we submitted a query such as "*What are the elapsed time and loss for training each epoch?*", the result is presented in Table 5.13. With this query, we can investigate, for example, if any epoch is taking longer than usual. In addition, with the loss value attribute selected along with the epoch identification, we can verify whether this value is improving over the epochs. Moreover, this query helps decision-making, such as tuning the optimizer or the learning rate.

Table 5.13 shows the current training epochs, time, and parameters. Based on Table 5.13, we decide on a further evaluation, considering training with two hyperparameter combinations for the PINN, executing in parallel, one with the optimizer Adam and the other with the optimizer RMSProp, both with the activation function Sigmoid. Then, we submit another query to compare the two optimizers: "*What is the elapsed time and training loss in the latest epoch for each combination?*", with the results shown in Table 5.14. From this query, we observe that Adam's optimizer loss presents the best results. Therefore, we decided to stop the training with RMSProp. Consequently, the relationship between the provenance data and the DL workflow configuration can help us tune the hyperparameter configurations during the search for the most satisfactory configuration.

Table 5.13: Query: "What are the elapsed time and loss for each training epoch?"

Epoch	**Time (s)**	$\mathcal{L}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{D}}$
0	3.157	1191.801	0.0122093	0.5859	47.64812
10000	2.194	0.016	0.0000047	0.0015	0.00058
20000	2.125	0.011	0.0000013	0.0004	0.00043
30000	2.347	0.025	0.0000018	0.0011	0.00097

Table 5.14: Query: "What is the elapsed time and training loss in the latest epoch for each combination?"

Time (s)	$\mathcal{L}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{D}}$	**Optimizer**	**Act Func**
2.193	0.438	0.0000392	0.0009	0.017	RMSProp	Sigmoid
2.169	0.013	0.0000004	0.0008	0.0005	Adam	Sigmoid

In the second experiment, conducted on SDumont, the user assigned values for the hyperparameters based on past experiments. The hyperparameters batch size = 3,000, number of epochs = 400,000, initial learning rate = 0.001, and final learning rate = 0.000018 were used with the exponential decay function, with decay = 0.99. DLProv for PINNs captures the provenance of the training output information for each configuration, persisting the data in the database at an interval of 100 epochs. The experiment was divided into two rounds that evaluated five and four configurations, respectively. In the first round, the user explores different optimizers and activation functions for each neural network, whereas in the second round, the user varies the number of neurons and intermediate layers. The idea is that at the end of each round, the user tunes the hyperparameters to improve the performance of the model evaluation metric R^2 and the loss function. During each round, the user queries the provenance database to assess the performance of each configuration. They can aggregate data that complements visual analyses and yet is not trivial to be examined in typical graphical tools like TensorBoard. The following queries show a few examples.

PINN-Q1: What are the configurations for the trained PINNs?

PINN-Q2: What are the largest values of R^2, and what is the epoch and time taken to obtain them?

PINN-Q3: What are the lowest values of the loss function and its components, what is the epoch, and the time it took?

PINN-Q1 shows some of the hyperparameters that are being used in the two networks to train the PINN for each configuration. PINN-Q2 and PINN-Q3 assist the user in identifying the highest R^2 and lowest loss values, respectively, obtained during the training, as well as the time taken to reach these values. These queries can help to analyze the trade-off between training time and performance of the configurations, so they can drop or adjust the configurations for the next round, or even abort the execution, since the queries can be submitted during the training.

By analyzing the first round results presented in Tables 5.15, 5.16, 5.17, along with Figures 5.21 and 5.22, the user can gain a few insights. For instance, PINN-Q2 highlights that configuration 3 took almost as much time as configuration 5 but produced a much lower

R^2. This may indicate that Adam with Sigmoid could be discarded in future rounds. PINN-Q3 shows details of the PINN loss, which is the main metric for inverse problems.

Table 5.15: PINN-Q1 results with the configurations for the training

ID	Optimizer	Activation TT	Activation Vel	Neurons TT	Neurons Vel	Intermediate Layers TT	Intermediate Layers Vel
1	Adam	tanh	tanh	20	32	8	4
2	Adam	relu	relu	20	32	8	4
3	Adam	sigmoid	sigmoid	20	32	8	4
4	Adam	relu	tanh	20	32	8	4
5	RMSProp	tanh	tanh	20	32	8	4

Table 5.16: PINN-Q2 results ordered by the largest values of R^2 with its epoch and time taken to obtain them

ID	Max R^2	Epoch	Time (m)
5	0.470	398600	100.912
1	0.377	88000	22.293
3	0.105	397200	100.756
2	0.093	86300	20.165
4	0.077	3400	0.790

Table 5.17: PINN-Q3 results ordered by the lowest values of the loss function and its components, with its epoch and time taken

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Epoch	Time (m)
5	$4.503 \cdot 10^{-5}$	$3.727 \cdot 10^{-5}$	$4.066 \cdot 10^{-10}$	$3.102 \cdot 10^{-7}$	399900	101.241
1	$1.283 \cdot 10^{-4}$	$3.692 \cdot 10^{-5}$	$3.771 \cdot 10^{-8}$	$3.654 \cdot 10^{-6}$	398100	100.852
4	$2.229 \cdot 10^{-4}$	$1.252 \cdot 10^{-4}$	$7.298 \cdot 10^{-9}$	$3.910 \cdot 10^{-6}$	359500	83.584
2	$3.290 \cdot 10^{-4}$	$1.677 \cdot 10^{-4}$	$1.527 \cdot 10^{-9}$	$6.449 \cdot 10^{-6}$	364000	85.055
3	$3.636 \cdot 10^{-3}$	$5.861 \cdot 10^{-4}$	$2.212 \cdot 10^{-8}$	$1.220 \cdot 10^{-4}$	343700	87.185

In the second round, the user maintains the hyperparameters related to configurations 1 and 5, but changes the topology of the TT and Vel networks, doubling the number of intermediate layers and adjusting the neurons for each one of them.

When analyzing Tables 5.18 and 5.19, with Figures 5.23 and 5.24, the user can assess that, overall, changing the number of intermediate layers and neurons did not improve the results of the previous round. Although configuration 8 presented an R^2 of 0.380, which is greater than the R^2 of configuration 1, configuration 8 took a considerably longer time ($\approx$ 74 minutes). On the other hand, the loss function achieved lower values (Table 5.20). Based on these analyses, one can also notice that as the complexity of the PINN increases, the loss function tends to be smaller. However, this reduction does not follow the improvement in the predictive capacity of the model given by the metric R^2, indicating the need for new configurations.

The user can further tune the workflow configurations to explore better PINN models. By submitting query PINN-Q3, the user can identify that the best model was configuration

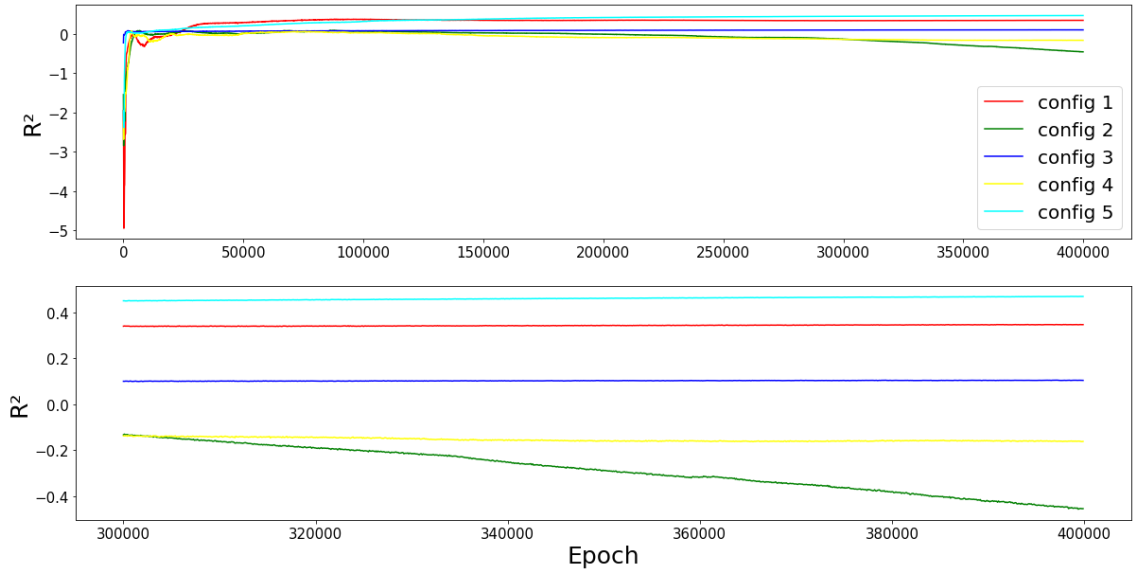


Figure 5.21: Metric value R^2 for all epochs (upper); Focus on the last 100,000 epochs (lower). Regarding the first round.

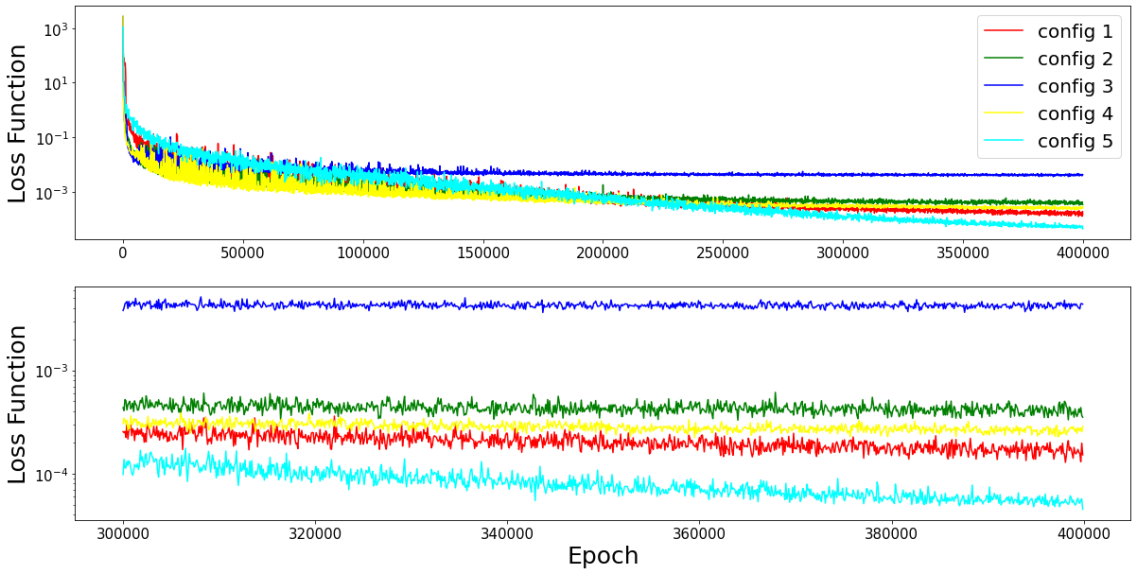


Figure 5.22: Loss function value for all epochs (Upper); Focus on the last 100,000 epochs (Lower). Regarding the first round.

Table 5.18: PINN-Q1 results with configurations of the second round

ID	Optimizer	Activation TT	Activation Vel	Neurons TT	Neurons Vel	Intermediate Layers TT	Intermediate Layers Vel
6	Adam	tanh	tanh	40	64	16	8
7	RMSProp	tanh	tanh	40	64	16	8
8	Adam	tanh	tanh	20	32	16	8
9	RMSProp	tanh	tanh	20	32	16	8

Table 5.19: PINN-Q2 results ordered by the largest values of R^2, with its epoch and time in the second round

ID	Max R^2	Epoch	Time (m)
8	0.380	399800	174.046
6	0.219	8200	3.664
7	0.204	37400	17.011
9	0.194	124100	54.914

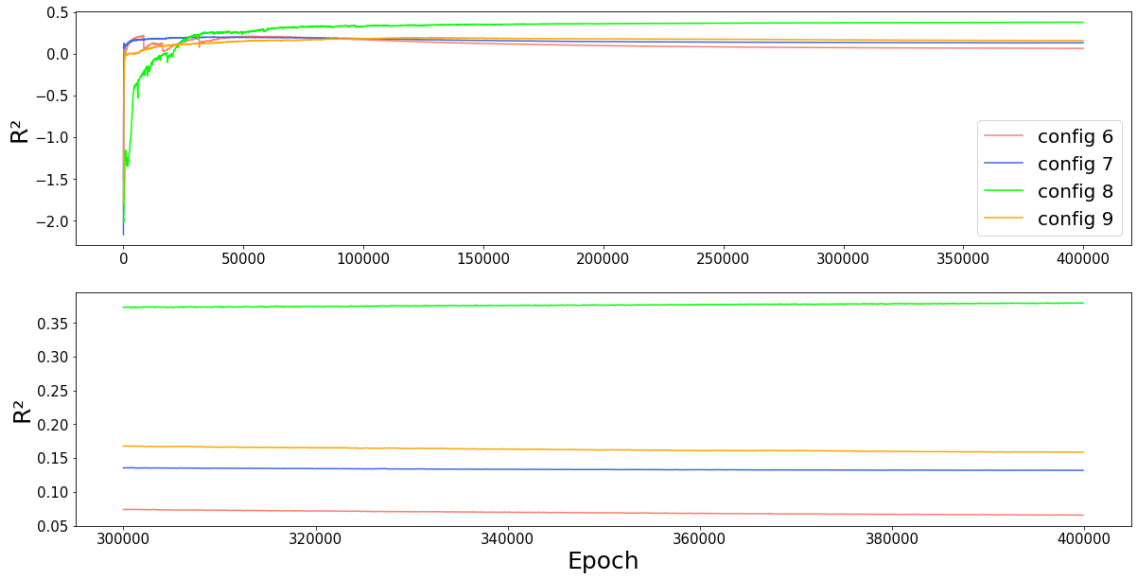


Figure 5.23: Metric value R^2 for all epochs (upper); Focus on the last 100,000 epochs (lower). Regarding the second round.

Table 5.20: PINN-Q3 results ordered by the lowest values of the loss with its components, its epoch, and time in the second round

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Epoch	Time (m)
6	$2.919 \cdot 10^{-6}$	$2.866 \cdot 10^{-6}$	$1.182 \cdot 10^{-10}$	$2.130 \cdot 10^{-9}$	399700	78.599
7	$6.568 \cdot 10^{-6}$	$5.213 \cdot 10^{-6}$	$3.841 \cdot 10^{-10}$	$5.419 \cdot 10^{-8}$	395100	79.705
8	$7.503 \cdot 10^{-6}$	$6.897 \cdot 10^{-6}$	$4.287 \cdot 10^{-9}$	$2.408 \cdot 10^{-8}$	392100	70.694
9	$3.415 \cdot 10^{-5}$	$2.719 \cdot 10^{-5}$	$4.834 \cdot 10^{-10}$	$2.783 \cdot 10^{-7}$	397200	75.761

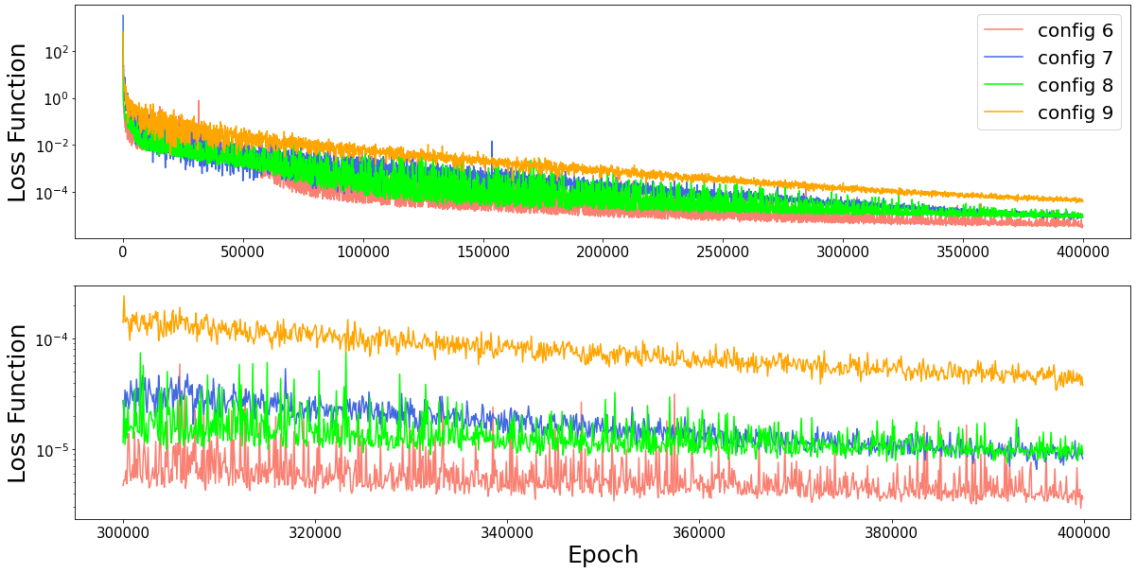
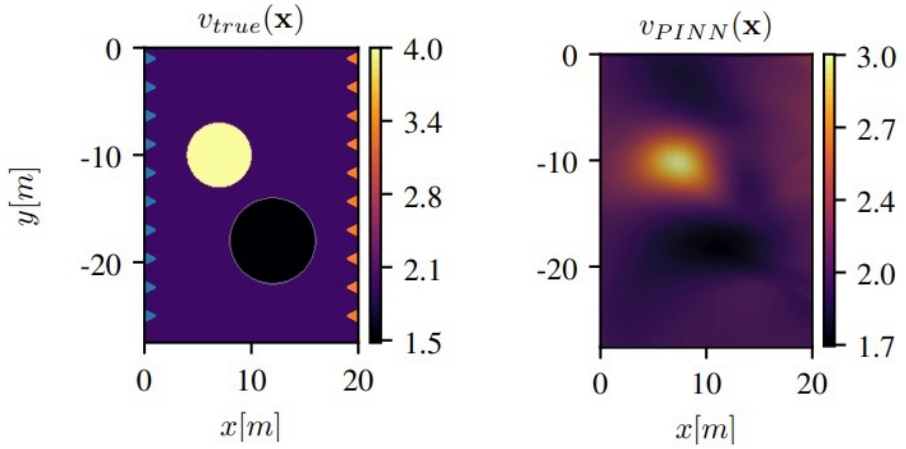


Figure 5.24: Loss function value for all epochs (Upper); Focus on the last 100,000 epochs (Lower). Regarding the second round.

5 with the lowest loss value for all its components, despite the metric $R^2 = 0.47$. This result is similar to the value obtained in SILVA *et al.* (2021b), which was validated with the best numerical method. Figure 5.25b shows how the prediction made by the PINN model generated with configuration 5 approaches the real solution (Figure 5.25a).



(a) Ground truth (b) PINN model

Figure 5.25: Comparison of prediction. **(a)** The ground truth velocity model corresponds to a background velocity model with $v_{true}(\mathbf{x}) = 2.0[km/s]$ with two inclusions with $v_{true}(\mathbf{x}) = 4.0[km/s]$ (top-left) and $v_{true}(\mathbf{x}) = 1.5[km/s]$ (bottom-right) SILVA *et al.* (2021b). **(b)** Prediction of the best PINN model with $R^2 = 0.47$.

If the user did not use the support of DLProv for PINNs, they would have to perform the management and capture of data referring to the DL workflow configuration and results of each model using log files, which would not provide the benefits of persistence of data made from DLProv for PINNs. They would also have to define an organization to store

these files, which in turn would not respect a predefined standard. In addition, they would have to write another script to associate the different log files referring to each model to carry out their analyses.

Overhead Analysis

DLProv can also be connected to data visualization tools, such as Kibana[17], to create dashboards and other resources to improve the runtime data analysis. Figure 5.26 shows the training loss of different executions, presenting an overall perspective of each run. From Figure 5.26, we observe that the configuration with RMSProp and Sigmoid shows the worst results. Continuing the training until the loss decreases to 10^{-3}, we see that it is a good decision to stop the training earlier for this configuration.

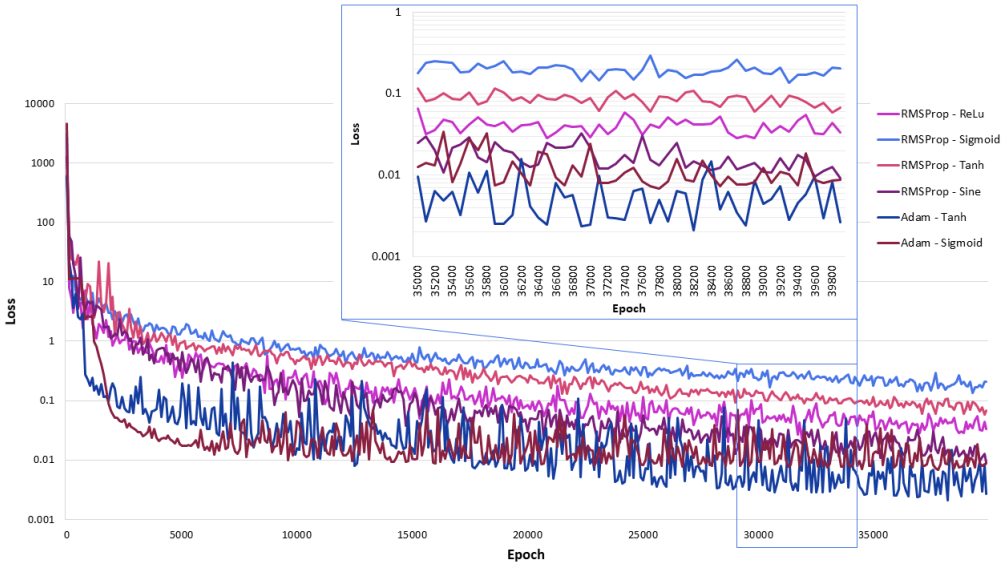


Figure 5.26: Graphical view of the training loss

Provenance capture introduces overhead on the PINN execution. We measured this overhead for all the runs during our initial hands-on experience on Grid5000, as shown in Figure 5.26. The overhead corresponds to an increase of 4% over the total time, which is considered negligible, given the DL workflow configuration tuning benefits. Such low overhead is due to the CPU-GPU hybrid-computing model, where the provenance management engine runs on the CPU and the PINN on the GPU. For comprehensive details and insights into these experiments, please refer to (SILVA *et al.*, 2021b).

[17]https://www.elastic.co/kibana

5.3.7 PINN Poisson and DeepXDE

This experiment further explores DLProv's independence from specific DL frameworks by using the DeepXDE framework to solve the inverse problem for the Poisson equation with an unknown forcing field[18]. This type of equation is often found in Physics and engineering problems. The equation is in one dimension and is of the form shown in Equation 5.3 and with the Dirichlet boundary conditions as presented in Equation 5.4.

$$\frac{d^2u(x)}{dx^2} = q(x), x \in [-1, 1] \tag{5.3}$$

$$u(-1) = u(1) = 0 \tag{5.4}$$

In this Poisson example, $u(x)$ and $q(x)$ are unknown. To solve the problem, the value of $u(x)$ is set to 100 points. The reference solution is $u(x) = sin(\pi x), q(x) = -\pi^2 sin(\pi x)$. Similarly to the Eikonal PINN, DeepXDE defines two networks for this Poisson example, one to train $u(x)$ and the other to train $q(x)$. The loss function $\mathcal{L}$ was also defined with three components related to $\mathcal{L}_{\mathcal{R}}$ as the PDE residual, $\mathcal{L}_{\mathcal{BC}}$ as boundary and initial conditions, and $\mathcal{L}_{\mathcal{D}}$ the data assimilation. DLProv for PINNs persists provenance data at every 10 epochs. We show results for 50000 epochs. The $L2$ relative error is used as an evaluation metric.

To conduct the comparative analysis of the Poisson PINN models, we submitted queries to the DLProv database, similar to the ones defined in the Eikonal PINN. These experiments were also performed on Santos Dumont. Table 5.21 shows the results of query PINN-Q1 with some attributes for the three configurations evaluated. We defined alternative combinations for the three weights of the $\mathcal{L}$ components. The hyperparameters referring to the neural network are the same for the two networks responsible for the estimates of $u(x)$ and $q(x)$.

Table 5.21: PINN-Q1 results with the training configurations

ID	**Optimizer**	**Activation**	**Neurons**	**Intermediate Layers**	**Learning Rate**	**Loss Weights** (w_R,w_{BC},w_D)
1	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(10,100,1000)
2	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(15,150,1500)
3	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(20,200,2000)

Note that this was a straightforward PINN training run, with each configuration taking no more than 90 seconds to complete 50,000 epochs. Queries PINN-Q2 and PINN-Q3 provide an analysis of the best $L2$ relative error metric and the lowest value of the loss

[18] `https://deepxde.readthedocs.io/en/latest/demos/pinn_inverse.html`

function for the three model configurations. Table 5.22 may indicate that configuration 3 has an advantage over the others, given the relative $L2$ error. The aggregated maximum and minimum query results provide additional insight into the trends shown in the graphical figures. In Figure 5.27, we can see that configurations 1 and 3 converged to the $L2$ relative error early in training, before epoch 5,000, and did not improve significantly after that. The behavior of the loss function during training, as depicted in Figure 5.28, indicates improvements over the epochs. However, these gains are not reflected by a significant change in the $L2$ metric. Table 5.23 presents the individual loss components across the different workflow configurations. Based on the relative $L2$ error, configuration 3 yields the most "satisfactory" model.

Table 5.22: PINN-Q2 results ordered by the lowest relative error values $L2$ for the function $u(x)$, with its epoch and time taken

ID	**Minimum** $L2$ relative error	**Epoch**	**Time (s)**
3	$1.98{\cdot}10^{-4}$	49590	83.064
2	$2.89{\cdot}10^{-4}$	49990	83.778
1	$2.97{\cdot}10^{-4}$	49860	88.809

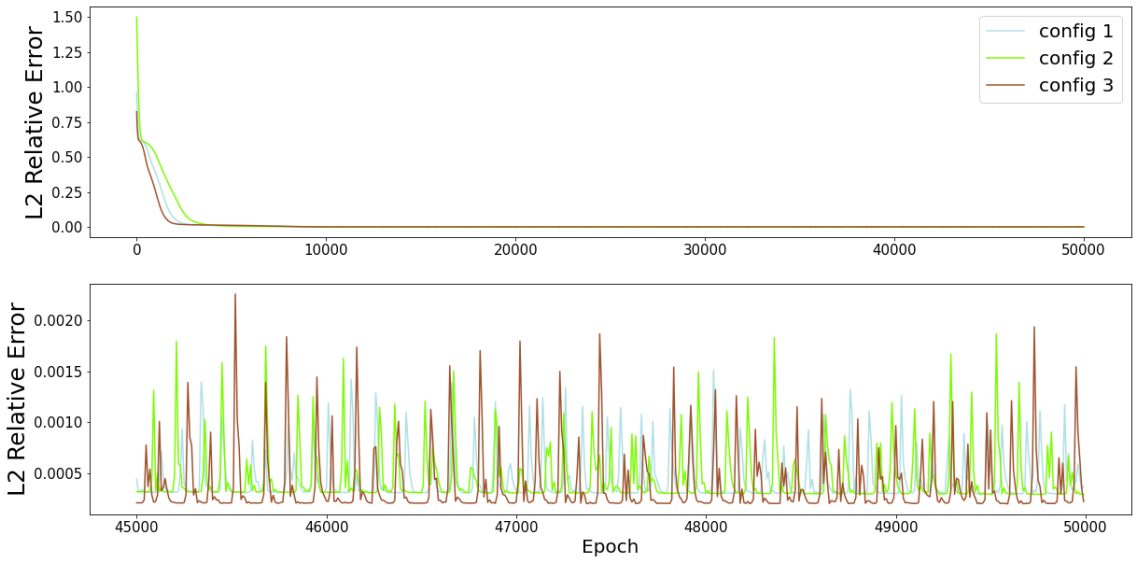


Figure 5.27: Relative error value $L2$ for function estimates $u(x)$ for all epochs (upper); Focus on the last 5,000 epochs (lower).

Table 5.23: PINN-Q3 results ordered by the lowest values of the loss with its components, its epoch, and time taken

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	**Epoch**	**Time(s)**
1	$5.384{\cdot}10^{-5}$	$9.980{\cdot}10^{-7}$	$1.975{\cdot}10^{9}$	$4.366{\cdot}10^{-8}$	49860	88.809
3	$6.359{\cdot}10^{-5}$	$1.017{\cdot}10^{-6}$	$2.037{\cdot}10^{8}$	$1.959{\cdot}10^{-8}$	49830	83.466
2	$6.751{\cdot}10^{-5}$	$1.666{\cdot}10^{-7}$	$1.166{\cdot}10^{8}$	$4.218{\cdot}10^{-8}$	49990	83.778

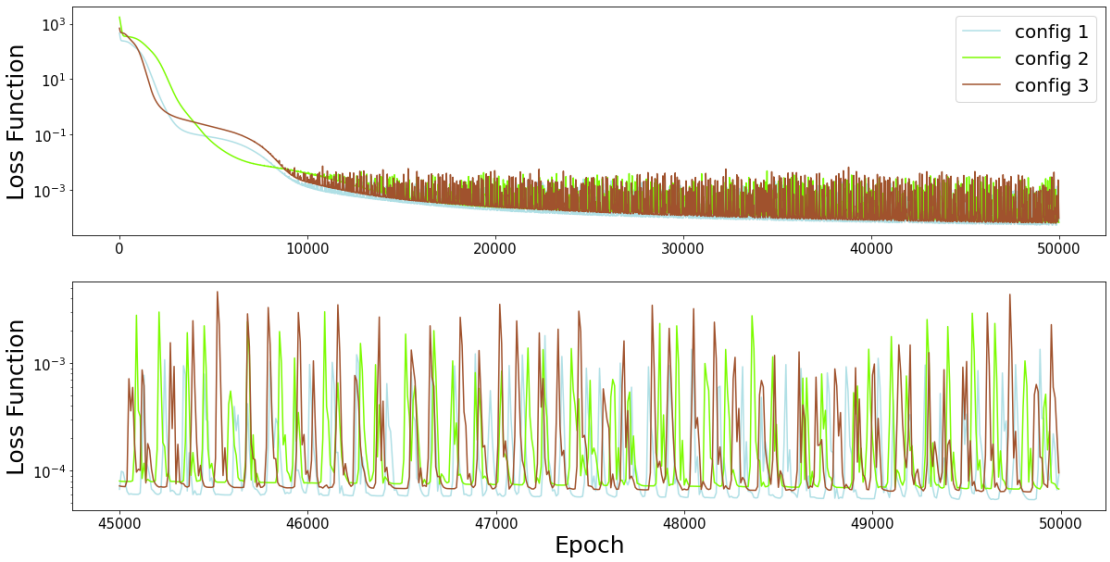


Figure 5.28: Loss function value for all epochs (Upper); Focus on the last 5,000 epochs (Lower).

5.4 Provenance Graph in Production

The focus of this experiment is to show how a provenance document can be used in a production environment, highlighting the importance of addressing RQ4. Queries to the provenance graph can play an important role in supporting the identification of discrepancies between the production data and training data. By leveraging the provenance graph, data scientists can retrieve and analyze, for instance, statistical measures from the training data, such as the distribution of categorical variables, the mean and standard deviation of numerical attributes, and other metrics relevant for evaluating the data's consistency. This ensures that relevant details about the training dataset are accessible and can be queried to compare against the production data.

Going back to the example from Section 5.1 where a DL model has been trained on the Framingham Heart Disease dataset (FHS) dataset, which includes several risk factors like age, cholesterol levels, blood pressure, smoking habits, and diabetes status to predict the risk of heart disease. The training data consists of individuals aged between 30 and 70, with the majority of data points coming from individuals in their 50s and 60s, as this is the typical age group affected by cardiovascular disease.

When the DL model is put into production, it starts receiving data from a healthcare application monitoring younger individuals, mostly in their 20s and 30s, for heart disease risk. This population is significantly underrepresented in the Framingham training dataset. The production data includes younger individuals who may not display the typical risk

factors (like high cholesterol or blood pressure) at first glance but could still be at risk due to genetic factors, lifestyle choices, or early signs of heart disease.

The model trained on the FHS data, which predominantly covers middle-aged adults, may not generalize well to the younger population in the production data. If the production data ages are consistently below the ages found in the training data, the model may carry over assumptions, such as a higher likelihood of heart disease in older individuals, leading to an underestimation of risk in younger individuals. Additionally, the relationship between risk factors and heart disease in the training data may not apply in the same way to younger individuals, as the Framingham model was not trained on this younger population. By not accurately assessing the risk of heart disease in younger individuals, the model may fail to flag high-risk cases that could have been treated early. This could result in poor health outcomes for individuals in their 20s and 30s, who may not have been prioritized for screening in the original dataset.

A good starting point for investigating this issue is to examine the provenance graph associated with the DL model in production. This includes analyzing how the data was preprocessed, understanding the distribution of the training dataset, and tracking key transformations from the provenance graph provided together with the DL model. In addition, from the *DL Model Provenance* database, a scientist can retrieve the `record_id` for each tuple used to train or evaluate the DL model. Using these retrieved values of `record_id`, the scientist can then query the *Preprocessing Provenance* database to obtain attribute values, such as age, associated with these tuples and verify the preprocessing operations applied to the data. These queries can be executed directly in MongoDB (the DBMS where the *Preprocessing Provenance* database is stored), via Python using *pymongo*[19], or through Apache Drill. Figure 5.29 presents an example of the provenance graph for the training of the FHS dataset. The gray area highlights a subset of the provenance stored in the *Preprocessing Provenance* database, representing final preprocessed entities at a fine level of granularity. The remaining entities and activities are retrieved from the *DL Model Provenance* database, providing a broader view of the model's provenance.

In addition to retrieving individual attribute values, queries can be designed to extract attribute distributions from the training dataset. Comparing these distributions with those observed in production helps identify potential data drifts in data feature distributions over time. For instance, Figure 5.30 illustrates the distribution of the `age` attribute in the training dataset, retrieved from the provenance graph. By comparing it with the corresponding distribution in production, scientists can detect discrepancies that may indicate shifts in the population served by the model. Such changes can signal the need for retraining or tuning to ensure the model's continued reliability.

[19] `https://pymongo.readthedocs.io/en/stable/`

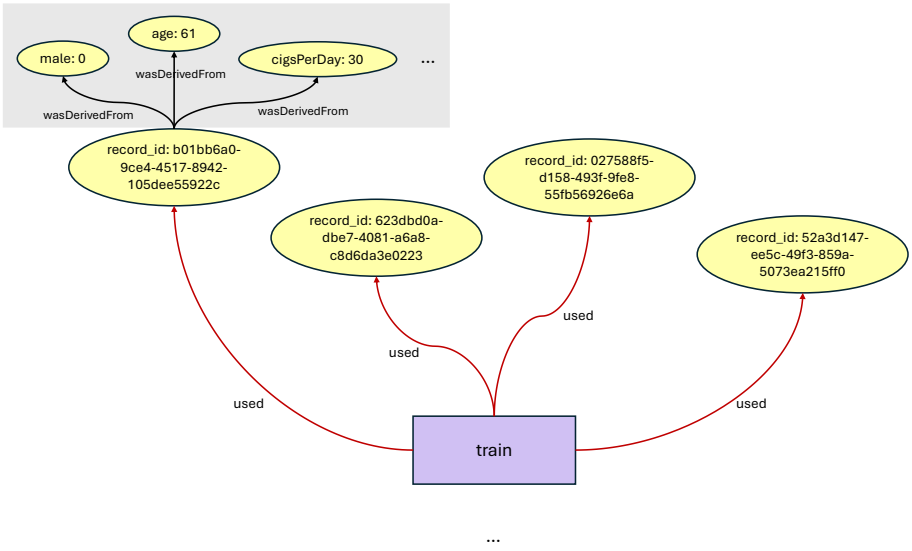


Figure 5.29: Provenance graph of the training process for the FHS dataset. The gray area highlights a fragment stored in the *Preprocessing Provenance* database, while the remaining elements come from the *DL Model Provenance* database.

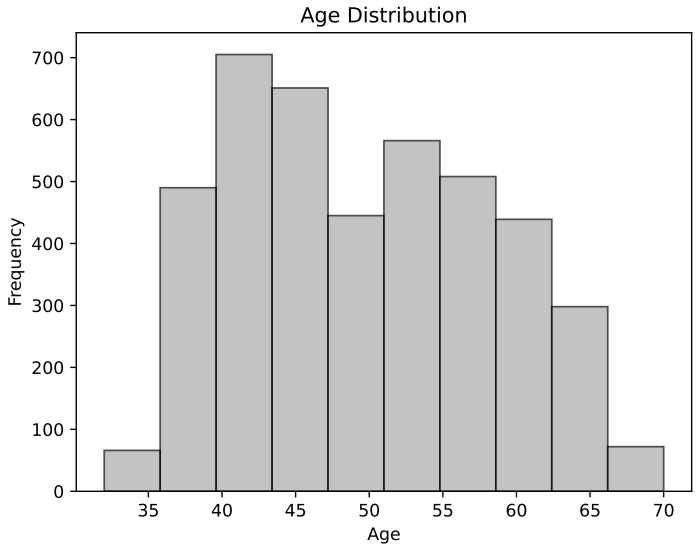


Figure 5.30: Age distribution in the training dataset, retrieved from the provenance graph linked to the DL model in production.

Chapter 6

Final Remarks

This chapter presents the final considerations of this thesis, summarizing its main contributions, discussing its limitations, and outlining directions for future work. The Conclusion section revisits the research contributions and highlights the experiments performed to validate them, showing how DLProv addresses key challenges in DL workflows' traceability. The Scope and Limitations section defines the boundaries of the work and acknowledges areas where DLProv may fall short. Finally, the Future Work section discusses potential research directions.

6.1 Conclusions

Supporting traceability of DL workflows is important for enabling thorough analysis during the generation and selection of DL models. This helps improve decision-making by helping data scientists identify which aspects of the workflow should be tuned and which model should ultimately be deployed. Moreover, traceability plays a key role in ensuring reproducibility and building trust in deployed DL models. By capturing detailed information about the DL model, including DL workflow configuration (the data and hyperparameters used, the preparation steps, and other relevant data) and DL workflow results, traceability ensures that the process is transparent and that DL model scientists can trust the model's performance and reliability.

In this thesis, we introduced DLProv, a suite of provenance services designed to allow traceability by capturing provenance across the steps of DL workflows and representing key relationships. To address RQ1, which concerns data derivation traces fragmented across different workflow steps, we ensured traceability by integrating provenance across the DL workflow, providing a connected and comprehensive view. To overcome the limitations of ad-hoc provenance representations raised in RQ2, we offered an extensible provenance model that captures activities, agents, and entities specific to DL tasks while

following the W3C PROV as a standardized recommendation for provenance representation. Acknowledging the tightly coupled nature of many provenance solutions raised in RQ3, we designed a flexible provenance capture mechanism that operates across arbitrary execution environments. This allows provenance services to be invoked independently of the DL framework through instrumentation. We also offered instances as specialized implementations, such as DLProv for Keras and DLProv for PINNs, tailored to track provenance in the Keras library and PINNs, respectively. To address the absence of independent provenance documents highlighted in RQ4, we offered DLProv support to provenance graph generation following the W3C PROV model in multiple formats, including JSON and PROV-N. Furthermore, we supported their ingestion into Neo4j, facilitating interoperability and enabling post-deployment analysis.

To validate traceability, we conducted experiments that integrate provenance from preprocessing, captured by an early version of DPDS, with the provenance of DLProv for DL model training and evaluation. We submitted a set of queries gathered from the literature to analyze how provenance from these steps can be combined. Additionally, we explored how Weights & Biases supports traceability, and we compared the traceability provided by MLflow and MLflow2PROV against that of DLProv. These experiments showed how provenance queries can relate input data, hyperparameters, and evaluation metrics, offering insights into the impact of preprocessing and configuration decisions across different steps of the DL workflow on model performance.

To validate DLProv's ability to operate in arbitrary execution environments, we conducted experiments using several DL frameworks, including TensorFlow, Keras, PyTorch, and DeepXDE. We also tested DLProv in diverse computational environments, such as simple laptops, Google Colab, and high-performance computing resources like Lobo Carneiro, Grid5000, and Santos Dumont. These experiments showed that DLProv can capture and analyze provenance across multiple DL scenarios, DL frameworks, and computational environments, highlighting its flexibility.

To validate the generation of an independent provenance document, we ran experiments simulating the decision to deploy a DL model and then generated a provenance graph of this model. We further showed how this provenance graph can support drift detection in a production environment by analyzing the provenance of the training process of the deployed model.

The provenance representation following W3C PROV was validated across all these experiments. We showed how provenance stored in the DL Model Provenance database during DL model training and selection in MonetDB could be queried, as well as how the generated provenance graph could be ingested and analyzed in Neo4j. These experiments showed that despite the use of different DL frameworks and computational environments,

the provenance data representation remained consistent, ensuring interoperability and enabling comparative analyses across different DL workflows. In addition, we explored DLProv's ability to adapt to different contexts, such as tracking inference, highlighting its flexibility in addressing a range of provenance-related challenges.

Finally, we evaluated the overhead introduced by DLProv's provenance capture. Our experiments showed that the additional computational cost is negligible, ensuring that DLProv can be seamlessly integrated into DL workflows without compromising workflow execution time.

The development of DLProv shows that it could function as a standalone service for provenance capture to provide traceability, embracing the concept of "provenance as a service". Rather than having each DL framework implement its isolated provenance solution, DLProv services could be embedded into these frameworks, promoting provenance as a first-class citizen in the DL ecosystem.

6.2 Scope and Limitations

DLProv is not recommended for small-scale experiments where DL workflows can be easily and quickly repeated, as provenance may not be a concern in such cases. In scenarios that require very fine-grained tracking, users should assess whether provenance capture is necessary, as the added overhead may outweigh its potential benefits.

DLProv design is intentionally generic, allowing flexibility across different DL workflows. Although DLProv services could be extended to other ML algorithms, it would require further development depending on the specific ML workflow. For example, supporting federated learning would require the capture service to handle distributed execution, along with extensions to the provenance model to reflect decentralized updates (LOPES *et al.*, 2024).

DLProv still has some limitations. Some instances of DLProv adopt a DL script instrumentation approach to allow for flexible provenance data capture for analysis, like MLflow, Weights & Biases[1], and ModelKB GHARIBI *et al.* (2019). Therefore, there is a trade-off between DLProv instrumentation and use. On one hand, the instrumentation may impose a burden on scientists who need to associate script variables with provenance attributes. For traditional hyperparameters and metrics, this burden (of making these associations and instrumenting the script) can be alleviated through a service that automatically parses the user's script, making these associations. Alternatively, if services like DLProv are adopted by popular ML frameworks such as TensorFlow, PyTorch, and Keras, and integrated into their libraries, instrumentation could become automatic. On the

[1] `https://wandb.ai/site`

other hand, when scientists undertake code instrumentation themselves, queries tend to be more intuitive, since they gain an understanding of what is being captured, their names, and relationships.

6.3 Future Work

For future work, we plan to explore provenance capture for the execution of DL workflows at different levels of granularity and bundle them together with their associated objects, enabling a more detailed representation of the workflow. By capturing provenance at multiple levels, we can provide greater flexibility for data scientists to analyze workflows at a high level or drill down into specific details as needed. DLProv would also benefit from a benchmark with queries to assess traceability, reproducibility, and transparency across different DL workflows.

Moreover, future work involves addressing the challenge of capturing provenance in continuum computing environments. Building on our experience with parallel environments using Parsl and edge computing (ROSENDO *et al.*, 2023), we plan to develop methods for capturing and managing provenance data in dynamic, real-time workflows.

Beyond supporting the detection of discrepancies, the provenance graph also plays an important role in mitigating bias when it is identified. After addressing the underlying issue or correcting the discrepancies, a new provenance graph can be generated to reflect the updated training process or new data preprocessing steps. This new graph can serve as a baseline for monitoring future occurrences of drift. If another drift event arises, the updated provenance records allow for a reevaluation of the data and enable comparisons with prior versions of the workflow. Thus, the provenance graph in a production environment not only supports the identification of discrepancies but can also provide a dynamic and traceable framework for continuous monitoring and adaptation in DL workflows.

Another direction for future work is using provenance data to support hyperparameter recommendation. DLProv already captures important elements for this task, including DL workflow configuration and results. The next step involves developing mechanisms to analyze the captured provenance data, identifying patterns and correlations, and drawing insights into the impact of different hyperparameter configurations on model performance. Since performance metrics can be a different measurement for each data scientist, it is important to study how to provide such a mechanism, considering the data scientist's decision. Then, recommendations based on these analyses can be provided. Additionally, recognizing the importance of human participation in the workflow, it is important to establish a feedback loop to continuously improve the hyperparameter recommendation system.

References

AGRAWAL, P., ARYA, R., BINDAL, A., BHATIA, S., GAGNEJA, A., GODLEWSKI, J., LOW, Y., MUSS, T., PALIWAL, M. M., RAMAN, S., SHAH, V., SHEN, B., SUGDEN, L., ZHAO, K., WU, M.-C., 2019, "Data Platform for Machine Learning". In: *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, p. 1803–1816, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450356435. doi: 10.1145/3299869.3314050. Disponível em: <https://doi.org/10.1145/3299869.3314050>.

ALMEIDA, R. F., DA SILVA, W. M. C., CASTRO, K., DE ARAÚJO, A. P. F., WALTER, M. E. T., LIFSCHITZ, S., HOLANDA, M., 2019, "Managing data provenance for bioinformatics workflows using AProvBio", *Int. J. Comput. Biol. Drug Des.*, v. 12, n. 2, pp. 153–170. doi: 10.1504/IJCBDD.2019.099761. Disponível em: <https://doi.org/10.1504/IJCBDD.2019.099761>.

BALOUEK, D., CARPEN AMARIE, A., CHARRIER, G., DESPREZ, F., JEANNOT, E., JEANVOINE, E., LÈBRE, A., MARGERY, D., NICLAUSSE, N., NUSSBAUM, L., RICHARD, O., PÉREZ, C., QUESNEL, F., ROHR, C., SARZYNIEC, L., 2013, "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: Ivanov, I. I., van Sinderen, M., Leymann, F., Shan, T. (Eds.), *Cloud Computing and Services Science*, v. 367, *Communications in Computer and Information Science*, Springer International Publishing, pp. 3–20. ISBN: 978-3-319-04518-4. doi: 10.1007/978-3-319-04519-1_1.

BARREDO ARRIETA, A., DÍAZ-RODRÍGUEZ, N., DEL SER, J., BENNETOT, A., TABIK, S., BARBADO, A., GARCIA, S., GIL-LOPEZ, S., MOLINA, D., BENJAMINS, R., CHATILA, R., HERRERA, F., 2020, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI", *Information Fusion*, v. 58, pp. 82–115. ISSN: 1566-2535. doi: https://doi.org/10.1016/j.inffus.2019.12.012.

Disponível em: <https://www.sciencedirect.com/science/article/pii/S1566253519308103>.

BARRINGER, H., GROCE, A., HAVELUND, K., SMITH, M., 2010, "Formal analysis of log files", *Journal of aerospace computing, information, and communication*, v. 7, n. 11, pp. 365–390.

BELHAJJAME, K., B'FAR, R., CHENEY, J., COPPENS, S., CRESSWELL, S., GIL, Y., GROTH, P., KLYNE, G., LEBO, T., MCCUSKER, J., MILES, S., MYERS, J., SAHOO, S., TILMES, C., 2013, "PROV-DM: The PROV Data Model", (April). Disponível em: <https://eprints.soton.ac.uk/356851/>.

BENGIO, Y., 2012, "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*, Springer, pp. 437–478.

BERGSTRA, J., BENGIO, Y., 2012, "Random search for hyper-parameter optimization", *J. Mach. Learn. Res.*, v. 13, n. null (fev.), pp. 281–305. ISSN: 1532-4435.

BOEHM, M., KUMAR, A., YANG, J., 2019, "Data management in machine learning systems", *Synthesis Lectures on Data Management*, v. 11, n. 1, pp. 1–173.

BRUNTON, S. L., KUTZ, J. N., 2019, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press. doi: 10.1017/9781108380690.

BUNEMAN, P., KHANNA, S., WANG-CHIEW, T., 2001, "Why and Where: A Characterization of Data Provenance". In: Van den Bussche, J., Vianu, V. (Eds.), *Database Theory — International conference on database theory (ICDT) 2001*, pp. 316–330, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN: 978-3-540-44503-6.

CASHMAN, D., HUMAYOUN, S. R., HEIMERL, F., PARK, K., DAS, S., THOMPSON, J., SAKET, B., MOSCA, A., STASKO, J. T., ENDERT, A., GLEICHER, M., CHANG, R., 2018, "Visual Analytics for Automated Model Discovery", *CoRR*, v. abs/1809.10782. Disponível em: <http://arxiv.org/abs/1809.10782>.

CEOLIN, D., GROTH, P., VAN HAGE, W. R., NOTTAMKANDATH, A., FOKKINK, W. J., 2012, "Trust Evaluation through User Reputation and Provenance Analysis." *URSW*, v. 900, pp. 15–26.

CHAI, C., WANG, J., LUO, Y., NIU, Z., LI, G., 2023, "Data Management for Machine Learning: A Survey", *IEEE Transactions on Knowledge and Data Engineering*, v. 35, n. 5, pp. 4646–4667. doi: 10.1109/TKDE.2022.3148237.

CHAI, J., ZENG, H., LI, A., NGAI, E. W., 2021, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios", *Machine Learning with Applications*, v. 6, pp. 100134. ISSN: 2666-8270. doi: https://doi.org/10.1016/j.mlwa.2021.100134. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666827021000670>.

CHAPMAN, A., MISSIER, P., SIMONELLI, G., TORLONE, R., 2020, "Capturing and Querying Fine-Grained Provenance of Preprocessing Pipelines in Data Science", *Proc. VLDB Endow.*, v. 14, n. 4 (dec), pp. 507–520. ISSN: 2150-8097. doi: 10.14778/3436905.3436911. Disponível em: <https://doi.org/10.14778/3436905.3436911>.

CHAPMAN, A., MISSIER, P., LAURO, L., TORLONE, R., 2022, "DPDS: Assisting Data Science with Data Provenance", *PVLDB*, v. 15, n. 12, pp. 3614 – 3617. doi: 10.14778/3554821.3554857. Disponível em: <https://vldb.org/pvldb/vol15/p3614-torlone.pdf>.

CHAPMAN, A., LAURO, L., MISSIER, P., TORLONE, R., 2024, "Supporting Better Insights of Data Science Pipelines with Fine-grained Provenance", *ACM Transactions on Database Systems*, (fev.). ISSN: 0362-5915. doi: 10.1145/3644385. Disponível em: <https://dl.acm.org/doi/10.1145/3644385>. Just Accepted.

CHEN, A., CHOW, A., DAVIDSON, A., DCUNHA, A., GHODSI, A., HONG, S. A., KONWINSKI, A., MEWALD, C., MURCHING, S., NYKODYM, T., OGILVIE, P., PARKHE, M., SINGH, A., XIE, F., ZAHARIA, M., ZANG, R., ZHENG, J., ZUMAR, C., 2020, "Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle". In: *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, DEEM'20, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450380232. doi: 10.1145/3399579.3399867.

CHENEY, J., CHITICARIU, L., TAN, W.-C., 2009, "Provenance in Databases: Why, How, and Where", *Foundations and Trends® in Databases*, v. 1, n. 4, pp. 379–474. ISSN: 1931-7883. doi: 10.1561/1900000006. Disponível em: <http://dx.doi.org/10.1561/1900000006>.

CLIFFORD, B., FOSTER, I., VOECKLER, J.-S., WILDE, M., ZHAO, Y., 2008, "Tracking provenance in a virtual data grid", *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 565–575.

CORRIGAN, D., CURCIN, V., ETHIER, J., FLYNN, A. J., SOTTARA, D., 2019, "Challenges of deploying Computable Biomedical Knowledge in real-world applications". In: *AMIA 2019, Washington, DC, USA, November 16-20, 2019*. AMIA. Disponível em: <http://knowledge.amia.org/69862-amia-1.4570936/t002-1.4575206/t002-1.4575207/3201770-1.4575319/3203261-1.4575316>.

CUOMO, S., DI COLA, V. S., GIAMPAOLO, F., ROZZA, G., RAISSI, M., PICCIALLI, F., 2022, "Scientific machine learning through physics–informed neural networks: Where we are and what's next", *Journal of Scientific Computing*, v. 92, n. 3, pp. 88.

DA SILVA, D. N. R., SIMÕES, A., CARDOSO, C., DE OLIVEIRA, D. E. M., RITTMEYER, J. N., WEHMUTH, K., LUSTOSA, H., PEREIRA, R. S., SOUTO, Y. M., VIGNOLI, L. E. G., SALLES, R., DE S. C. JR, H., ZIVIANI, A., OGASAWARA, E. S., DELICATO, F. C., DE FIGUEIREDO PIRES, P., DA CUNHA PEREIRA PINTO, H. L., MAIA, L., PORTO, F., 2019, "A Conceptual Vision Toward the Management of Machine Learning Models." In: Panach, J. I., Guizzardi, R. S. S., Claro, D. B. (Eds.), *ER Forum/-Posters/Demos*, v. 2469, *CEUR Workshop Proceedings*, pp. 15–27. CEUR-WS.org. Disponível em: <http://dblp.uni-trier.de/db/conf/er/erf2019.html#SilvaSCORWLPSVS19>.

DA SILVA, G., DE OLIVEIRA, D., ROSSETI, I., PAES, A., 2023a, "LHTR.br: em Busca de um Conjunto Anotado de Textos Manuscritos em Português". In: *Anais do V Dataset Showcase Workshop*, pp. 13–24, Porto Alegre, RS, Brasil, a. SBC. doi: 10.5753/dsw.2023.233618. Disponível em: <https://sol.sbc.org.br/index.php/dsw/article/view/25501>.

DA SILVA, R. F., BADIA, R. M., BALA, V., et al., 2023b. "Workflows Community Summit 2022: A Roadmap Revolution". mar.b. Disponível em: <https://doi.org/10.5281/zenodo.7750670>.

DAVIDSON, S. B., FREIRE, J., 2008, "Provenance and Scientific Workflows: Challenges and Opportunities". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, p. 1345–1350, New York, NY, USA. Association for Computing Machinery. ISBN: 9781605581026. doi: 10.1145/1376616.1376772.

DAVISON, A., 2012, “Automated capture of experiment context for easier reproducibility in computational research”, *Computing in Science & Engineering*, v. 14, n. 4, pp. 48–56.

DE ALMEIDA, V. K., DE OLIVEIRA, D. E., DE BARROS, C. D. T., SCATENA, G. D. S., QUEIROZ FILHO, A. N., SIQUEIRA, F. L., COSTA, A. H. R., GOMI, E. S., MENDOZA, L. A. F., BATISTA, E. C. S., MUÑOZ, C. E., SIQUEIRA, I. G. A., BARREIRA, R. A., DOS SANTOS, I. H. F., CARDOSO, C., OGASAWARA, E., PORTO, F., 2024, “A Digital Twin System for Oil And Gas Industry: A Use Case on Mooring Lines Integrity Monitoring”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion ’24, p. 322–331, New York, NY, USA. Association for Computing Machinery. ISBN: 9798400706226. doi: 10.1145/3652620.3688244. Disponível em: <https://doi.org/10.1145/3652620.3688244>.

DE BIE, T., DE RAEDT, L., HERNÁNDEZ-ORALLO, J., HOOS, H. H., SMYTH, P., WILLIAMS, C. K. I., 2022, “Automating Data Science”, *Commun. ACM*, v. 65, n. 3 (feb), pp. 76–87. ISSN: 0001-0782. doi: 10.1145/3495256. Disponível em: <https://doi.org/10.1145/3495256>.

DE OLIVEIRA, L. S., KUNSTMANN, L., PINA, D., DE OLIVEIRA, D., MATTOSO, M., 2023, “PINNProv: Provenance for Physics-Informed Neural Networks”. In: *2023 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pp. 16–23. doi: 10.1109/SBAC-PADW60351.2023.00013.

DE OLIVEIRA, L. S., 2023, *GERÊNCIA DE DADOS DE PROVENIÊNCIA PARA AS REDES NEURAIS GUIADAS PELA FÍSICA*. Tese de Doutorado, Universidade Federal do Rio de Janeiro.

DEBNATH, L., 2012, “First-Order Nonlinear Equations and Their Applications”. In: *Nonlinear Partial Differential Equations for Scientists and Engineers*, pp. 227–256, Boston, Birkhäuser Boston. ISBN: 978-0-8176-8265-1. doi: 10.1007/978-0-8176-8265-1_4. Disponível em: <https://doi.org/10.1007/978-0-8176-8265-1_4>.

DEELMAN, E., GANNON, D., SHIELDS, M., TAYLOR, I., 2009, “Workflows and e-Science: An overview of workflow system features and capabilities”, *Future Generation Computer Systems*, v. 25, n. 5, pp. 528–540. ISSN: 0167-739X. doi: https://doi.org/10.1016/j.future.2008.06.012.

Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X08000861>.

DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., FEI-FEI, L., 2009, "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

DROZDAL, J., WEISZ, J., WANG, D., DASS, G., YAO, B., ZHAO, C., MULLER, M., JU, L., SU, H., 2020, "Trust in AutoML: Exploring Information Needs for Establishing Trust in Automated Machine Learning Systems". In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI '20, p. 297–307, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450371186. doi: 10.1145/3377325.3377501. Disponível em: <https://doi.org/10.1145/3377325.3377501>.

FAIRWEATHER, E., WITTNER, R., CHAPMAN, M., HOLUB, P., CURCIN, V., 2020, "Non-repudiable provenance for clinical decision support systems", *CoRR*, v. abs/2006.11233. Disponível em: <https://arxiv.org/abs/2006.11233>.

FEKETE, J., FREIRE, J., RHYNE, T., 2020, "Exploring Reproducibility in Visualization", *IEEE Computer Graphics and Applications*, v. 40, n. 5, pp. 108–119. doi: 10.1109/MCG.2020.3006412. Disponível em: <https://doi.org/10.1109/MCG.2020.3006412>.

FERREIRA DA SILVA, R., BARD, D., CHARD, K., et al., 2024, "Workflows Community Summit 2024: Future Trends and Challenges in Scientific Workflows", , n. ORNL/TM-2024/3573. doi: 10.5281/zenodo.13844759.

FREIRE, J., KOOP, D., SANTOS, E., SILVA, C. T., 2008, "Provenance for Computational Tasks: A Survey", *Computing in Science and Engineering*, v. 10, n. 3, pp. 11–21. doi: 10.1109/MCSE.2008.79.

FREITAS, C., OLIVEIRA, L. S., SABOURIN, R., BORTOLOZZI, F., 2008, "Brazilian forensic letter database". In: *11th International workshop on frontiers on handwriting recognition, Montreal, Canada*.

FREITAS, R. S., BARBOSA, C. H., GUERRA, G. M., COUTINHO, A. L., ROCHINHA, F. A., 2020, "An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty", *arXiv preprint arXiv:2006.09550*.

FREITAS, R. S., BARBOSA, C. H., GUERRA, G. M., COUTINHO, A. L., ROCHINHA, F. A., 2021, "An encoder-decoder deep surrogate for reverse time migration in seismic imaging under uncertainty", *Computational Geosciences*, v. 25, n. 3, pp. 1229–1250.

GARCIA-MOLINA, H., ULLMAN, J. D., WIDOM, J., 2008, *Database Systems: The Complete Book*. 2nd ed. , Prentice Hall. ISBN: 978-0131873254.

GEHANI, A., TARIQ, D., 2012, "SPADE: Support for provenance auditing in distributed environments". In: *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 101–120. Springer.

GHARIBI, G., WALUNJ, V., ALANAZI, R., RELLA, S., LEE, Y., 2019, "Automated Management of Deep Learning Experiments". In: *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, DEEM'19, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450367974. doi: 10.1145/3329486.3329495. Disponível em: <https://doi.org/10.1145/3329486.3329495>.

GHARIBI, G., WALUNJ, V., NEKADI, R., MARRI, R., LEE, Y., 2021, "Automated end-to-end management of the modeling lifecycle in deep learning", *Empirical Software Engineering*, v. 26, pp. 1–33.

GHOSHAL, D., PLALE, B., 2013, "Provenance from log files: a BigData problem". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pp. 290–297.

GIL, Y., HONAKER, J., GUPTA, S., MA, Y., D'ORAZIO, V., GARIJO, D., GADEWAR, S., YANG, Q., JAHANSHAD, N., 2019, "Towards Human-Guided Machine Learning". In: *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19, p. 614–624, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450362726. doi: 10.1145/3301275.3302324. Disponível em: <https://doi.org/10.1145/3301275.3302324>.

GIL, Y., GARIJO, D., KHIDER, D., KNOBLOCK, C. A., RATNAKAR, V., OSORIO, M., VARGAS, H., PHAM, M., PUJARA, J., SHBITA, B., VU, B., CHIANG, Y.-Y., FELDMAN, D., LIN, Y., SONG, H., KUMAR, V., KHANDELWAL, A., STEINBACH, M., TAYAL, K., XU, S., PIERCE, S. A., PEARSON, L., HARDESTY-LEWIS, D., DEELMAN, E., FERREIRA DA SILVA, R., MAYANI, R., KEMANIAN, A. R., SHI, Y., LEONARD, L., PECKHAM, S., STOICA, M., COBOURN, K., ZHANG, Z., DUFFY, C., SHU, L., 2021, "Ar-

tificial Intelligence for Modeling Complex Systems: Taming the Complexity of Expert Models to Improve Decision Making", *ACM Transactions on Interactive Intelligent Systems*.

GLOROT, X., BORDES, A., BENGIO, Y., 2011, "Deep Sparse Rectifier Neural Networks". In: Gordon, G., Dunson, D., Dudík, M. (Eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, v. 15, *Proceedings of Machine Learning Research*, pp. 315–323, Fort Lauderdale, FL, USA, 11–13 Apr. PMLR. Disponível em: <https://proceedings.mlr.press/v15/glorot11a.html>.

GONZÁLEZ-BELTRÁN, A., MAGUIRE, E., SANSONE, S.-A., ROCCA-SERRA, P., 2014, "linkedISA: semantic representation of ISA-Tab experimental metadata", *BMC bioinformatics*, v. 15, pp. 1–15.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A., 2016, *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

GRAFBERGER, S., GROTH, P., STOYANOVICH, J., SCHELTER, S., 2022, "Data distribution debugging in machine learning pipelines", *The VLDB Journal*, v. 31, n. 5, pp. 1103–1126.

GREGORI, L., MISSIER, P., STIDOLPH, M., TORLONE, R., WOOD, A., 2024, "Design and Development of a Provenance Capture Platform for Data Science". In: *Procs. 3rd DATAPLAT workshop, co-located with ICDE 2024*, Utrecht, NL, maio. IEEE.

HAGHIGHAT, E., JUANES, R., 2021, "SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks", *Computer Methods in Applied Mechanics and Engineering*, v. 373, pp. 113552. ISSN: 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113552. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782520307374>.

HE, K., ZHANG, X., REN, S., SUN, J., 2016, "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

HE, X., ZHAO, K., CHU, X., 2021, "AutoML: A survey of the state-of-the-art", *Knowledge-Based Systems*, v. 212, pp. 106622. ISSN: 0950-7051. doi: https://doi.org/10.1016/j.knosys.2020.106622. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950705120307516>.

HERSCHEL, M., DIESTELKÄMPER, R., LAHMAR, H. B., 2017, "A survey on provenance: What for? What form? What from?" *The VLDB Journal*, v. 26, n. 6, pp. 881–906.

HOFFMANN, N., EBRAHIMI POUR, N., 2024, "A Low Overhead Approach for Automatically Tracking Provenance in Machine Learning Workflows". In: *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 567–573. doi: 10.1109/EuroSPW61312.2024.00092.

HOHMAN, F., KAHNG, M., PIENTA, R., CHAU, D. H., 2019, "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers", *IEEE Transactions on Visualization and Computer Graphics*, v. 25, n. 8, pp. 2674–2693. doi: 10.1109/TVCG.2018.2843369.

HOOS, H., LEYTON-BROWN, K., 2014, "An efficient approach for assessing hyperparameter importance". In: *International conference on machine learning*, pp. 754–762.

HUANG, G., LIU, Z., VAN DER MAATEN, L., WEINBERGER, K. Q., 2017, "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2261–2269. IEEE Computer Society. doi: 10.1109/CVPR.2017.243. Disponível em: <https://doi.org/10.1109/CVPR.2017.243>.

HUYNH, T. D., EBDEN, M., FISCHER, J., ROBERTS, S., MOREAU, L., 2018, "Provenance network analytics: an approach to data analytics using data provenance", *Data Mining and Knowledge Discovery*, v. 32, pp. 708–735.

HUYNH, T. D., STALLA, S., MOREAU, L., 2019, "Provenance-based explanations for automated decisions: final IAA project report", .

IDOWU, S., OSMAN, O., STRÜBER, D., BERGER, T., 2024, "Machine learning experiment management tools: a mixed-methods empirical study", *Empirical Software Engineering*, v. 29, n. 4, pp. 1–35.

IOFFE, S., SZEGEDY, C., 2015, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: Bach, F., Blei, D. (Eds.), *Proceedings of the 32nd International Conference on Machine Learning*, v. 37, *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul. PMLR. Disponível em: <https://proceedings.mlr.press/v37/ioffe15.html>.

KANWAL, S., KHAN, F. Z., LONIE, A., SINNOTT, R. O., 2017, "Investigating reproducibility and tracking provenance–a genomic workflow case study", *BMC bioinformatics*, v. 18, pp. 1–14.

KARMAKER ("SANTU"), S. K., HASSAN, M. M., SMITH, M. J., XU, L., ZHAI, C., VEERAMACHANENI, K., 2021, "AutoML to Date and Beyond: Challenges and Opportunities", *ACM Comput. Surv.*, v. 54, n. 8 (oct). ISSN: 0360-0300. doi: 10.1145/3470918. Disponível em: <https://doi.org/10.1145/3470918>.

KHAN, F. Z., SOILAND-REYES, S., SINNOTT, R. O., LONIE, A., GOBLE, C., CRUSOE, M. R., 2019, "Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv", *GigaScience*, v. 8, n. 11 (11). ISSN: 2047-217X. doi: 10.1093/gigascience/giz095. giz095.

KIDGER, P., LYONS, T., 2020, "Universal approximation with deep narrow networks". In: *Conf. on learning theory*, pp. 2306–2327. PMLR.

KLAMBAUER, G., UNTERTHINER, T., MAYR, A., HOCHREITER, S., 2017. "Self-Normalizing Neural Networks". .

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., 2012, "Imagenet classification with deep convolutional neural networks". In: *NeurIPS*, pp. 1097–1105.

KUMAR, A., NAKANDALA, S., ZHANG, Y., LI, S., GEMAWAT, A., NAGRECHA, K., 2021, "Cerebro: A Layered Data Platform for Scalable Deep Learning". In: *11th Annual Conference on Innovative Data Systems Research (CIDR'21)*.

KUNSTMANN, L., PINA, D., SILVA, F., PAES, A., VALDURIEZ, P., DE OLIVEIRA, D., MATTOSO, M., 2021, "Online Deep Learning Hyperparameter Tuning based on Provenance Analysis", *Journal of Information and Data Management*, v. 12, n. 5, pp. 396–414.

LANGER, M., OSTER, D., SPEITH, T., HERMANNS, H., KÄSTNER, L., SCHMIDT, E., SESING, A., BAUM, K., 2021, "What do we want from Explainable Artificial Intelligence (XAI)? – A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research", *Artificial Intelligence*, v. 296, pp. 103473. ISSN: 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103473. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0004370221000242>.

LECUN, Y., BENGIO, Y., HINTON, G., 2015, "Deep learning", *nature*, v. 521, n. 7553, pp. 436–444.

LEE, D. J.-L., MACKE, S., 2020, "A Human-in-the-loop Perspective on AutoML: Milestones and the Road Ahead", *IEEE Data Engineering Bulletin*.

LEO, S., CRUSOE, M. R., RODRÍGUEZ-NAVAS, L., SIRVENT, R., KANITZ, A., DE GEEST, P., WITTNER, R., PIREDDU, L., GARIJO, D., FERNÁNDEZ, J. M., COLONNELLI, I., GALLO, M., OHTA, T., SUETAKE, H., CAPELLA-GUTIERREZ, S., DE WIT, R., KINOSHITA, B. P., SOILAND-REYES, S., 2024, "Recording provenance of workflow runs with RO-Crate", *PLOS ONE*, v. 19, n. 9 (09), pp. 1–35. doi: 10.1371/journal.pone.0309210. Disponível em: <https://doi.org/10.1371/journal.pone.0309210>.

LI, L., NAKANDALA, S., KUMAR, A., 2021, "Intermittent human-in-the-loop model selection using cerebro: a demonstration", *Proceedings of the VLDB Endowment*, v. 14, n. 12, pp. 2687–2690.

LOPES, C., NUNES, A. L., BOERES, C., DRUMMOND, L. M. A., DE OLIVEIRA, D., 2024, "Provenance-Based Dynamic Fine-Tuning of Cross-Silo Federated Learning". In: Barrios H., C. J., Rizzi, S., Meneses, E., Mocskos, E., Monsalve Diaz, J. M., Montoya, J. (Eds.), *High Performance Computing*, pp. 113–127, Cham. Springer Nature Switzerland. ISBN: 978-3-031-52186-7.

LU, L., MENG, X., MAO, Z., KARNIADAKIS, G. E., 2021, "DeepXDE: A deep learning library for solving differential equations", *SIAM Review*, v. 63, n. 1, pp. 208–228. doi: 10.1137/19M1274067.

MA, X., ZHENG, J. G., GOLDSTEIN, J. C., ZEDNIK, S., FU, L., DUGGAN, B., AULENBACH, S. M., WEST, P., TILMES, C., FOX, P., 2014, "Ontology engineering in provenance enablement for the National Climate Assessment", *Environmental Modelling & Software*, v. 61, pp. 191–205. ISSN: 1364-8152. doi: https://doi.org/10.1016/j.envsoft.2014.08.002. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364815214002254>.

MARTINO, M. D., QUARATI, A., ROSIM, S., NAMIKAWA, L. M., 2021, "Documenting flooding areas calculation: A PROV approach", *International Journal of Metadata, Semantics and Ontologies*, v. 15, n. 1, pp. 50–59.

MASTERS, D., LUSCHI, C., 2018. "Revisiting Small Batch Training for Deep Neural Networks". Disponível em: <https://arxiv.org/abs/1804.07612>.

MATTOSO, M., WERNER, C., TRAVASSOS, G. H., BRAGANHOLO, V., OGASAWARA, E., OLIVEIRA, D., CRUZ, S., MARTINHO, W., MURTA, L., 2010, "Towards supporting the life cycle of large scale scientific experiments", *International Journal of Business Process Integration and Management*, v. 5, n. 1, pp. 79.

MATTOSO, M., DIAS, J., OCAÑA, K. A., OGASAWARA, E., COSTA, F., HORTA, F., SILVA, V., DE OLIVEIRA, D., 2015, "Dynamic steering of HPC scientific workflows: A survey", *Future Generation Computer Systems*, v. 46, pp. 100–113.

MCPHILLIPS, T., BOWERS, S., BELHAJJAME, K., LUDÄSCHER, B., 2015, "Retrospective provenance without a runtime provenance recorder". In: *7th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 15)*.

MELGAR, L. A., DAO, D., GAN, S., GÜREL, N. M., HOLLENSTEIN, N., JIANG, J., KARLAS, B., LEMMIN, T., LI, T., LI, Y., RAO, S. X., RAUSCH, J., RENGGLI, C., RIMANIC, L., WEBER, M., ZHANG, S., ZHAO, Z., SCHAWINSKI, K., WU, W., ZHANG, C., 2021, "Ease.ML: A Lifecycle Management System for Machine Learning". In: *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. Disponível em: <http://cidrdb.org/cidr2021/papers/cidr2021_paper26.pdf>.

MIAO, H., LI, A., DAVIS, L. S., DESHPANDE, A., 2016, "Modelhub: Towards unified data and lifecycle management for deep learning", *arXiv preprint arXiv:1611.06224*.

MIAO, H., LI, A., DAVIS, L. S., DESHPANDE, A., 2017a, "ModelHub: Deep Learning Lifecycle Management". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 1393–1394, a. doi: 10.1109/ICDE.2017.192.

MIAO, H., LI, A., DAVIS, L. S., DESHPANDE, A., 2017b, "Towards unified data and lifecycle management for deep learning". In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 571–582. IEEE, b.

MIDDLETON, S. E., LETOUZÉ, E., HOSSAINI, A., CHAPMAN, A., 2022, "Trust, regulation, and human-in-the-loop AI: within the European region", *Communications of the ACM*, v. 65, n. 4, pp. 64–68.

MISSIER, P., BELHAJJAME, K., CHENEY, J., 2013, "The W3C PROV family of specifications for modelling provenance metadata". In: *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 773–776.

MORA-CANTALLOPS, M., SÁNCHEZ-ALONSO, S., GARCÍA-BARRIOCANAL, E., SICILIA, M.-A., 2021, "Traceability for trustworthy ai: A review of models and tools", *Big Data and Cognitive Computing*, v. 5, n. 2, pp. 20.

MOREAU, L., GROTH, P., 2013, "Provenance: an introduction to PROV", *Synthesis Lectures on the Semantic Web: Theory and Technology*, v. 3, n. 4, pp. 1–129.

MOSQUEIRA-REY, E., HERNÁNDEZ-PEREIRA, E., ALONSO-RÍOS, D., BOBES-BASCARÁN, J., FERNÁNDEZ-LEAL, Á., 2023, "Human-in-the-loop machine learning: a state of the art", *Artif. Intell. Rev.*, v. 56, n. 4, pp. 3005–3054. doi: 10.1007/s10462-022-10246-w. Disponível em: <https://doi.org/10.1007/s10462-022-10246-w>.

MURTA, L., BRAGANHOLO, V., CHIRIGATI, F., KOOP, D., FREIRE, J., 2014, "noWorkflow: capturing and analyzing provenance of scripts". In: *International Provenance and Annotation Workshop*, pp. 71–83. Springer.

NAIR, V., HINTON, G. E., 2010, "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, p. 807–814, Madison, WI, USA. Omnipress. ISBN: 9781605589077.

NAMAKI, M. H., FLORATOU, A., PSALLIDAS, F., KRISHNAN, S., AGRAWAL, A., WU, Y., ZHU, Y., WEIMER, M., 2020, "Vamsa: Automated Provenance Tracking in Data Science Scripts". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, p. 1542–1551, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450379984. doi: 10.1145/3394486.3403205.

NASCIMENTO, E., CASANOVA, M. A., 2024, "Querying Databases with Natural Language: The use of Large Language Models for Text-to-SQL tasks". In: *Anais Estendidos do XXXIX Simpósio Brasileiro de Bancos de Dados*, pp. 196–201, Porto Alegre, RS, Brasil. SBC. doi: 10.5753/sbbd_estendido.2024.240552. Disponível em: <https://sol.sbc.org.br/index.php/sbbd_estendido/article/view/30793>.

NIGENDA, D., KARNIN, Z., ZAFAR, M. B., RAMESHA, R., TAN, A., DONINI, M., KENTHAPADI, K., 2022, "Amazon sagemaker model monitor: A system for real-time insights into deployed machine learning models". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3671–3681.

NILSBACK, M.-E., ZISSERMAN, A., 2006, "A visual vocabulary for flower classification". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, v. 2, pp. 1447–1454. IEEE.

OCAÑA, K. A. C. S., SILVA, V., DE OLIVEIRA, D., MATTOSO, M., 2015, "Data Analytics in Bioinformatics: Data Science in Practice for Genomics Analysis Workflows". In: *IEEE e-Science*, pp. 322–331. doi: 10.1109/eScience.2015.50. Disponível em: <https://doi.org/10.1109/eScience.2015.50>.

ONO, J. P., CASTELO, S., LOPEZ, R., BERTINI, E., FREIRE, J., SILVA, C., 2021, "PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines", *IEEE Transactions on Visualization and Computer Graphics*, v. 27, n. 2, pp. 390–400. doi: 10.1109/TVCG.2020.3030361.

ORMENISAN, A. A., ISMAIL, M., HARIDI, S., DOWLING, J., 2020, "Implicit provenance for machine learning artifacts", *Proceedings of MLSys*, v. 20.

PIMENTEL, J. F., FREIRE, J., MURTA, L., BRAGANHOLO, V., 2019, "A Survey on Collecting, Managing, and Analyzing Provenance from Scripts", *ACM Comput. Surv.*, v. 52, n. 3, pp. 47:1–47:38. doi: 10.1145/3311955. Disponível em: <https://doi.org/10.1145/3311955>.

PINA, D., KUNSTMANN, L., DE OLIVEIRA, D., VALDURIEZ, P., MATTOSO, M., 2021, "Provenance Supporting Hyperparameter Analysis in Deep Neural Networks". In: Glavic, B., Braganholo, V., Koop, D. (Eds.), *Provenance and Annotation of Data and Processes*, pp. 20–38, Cham. Springer International Publishing. ISBN: 978-3-030-80960-7.

PINA, D., CHAPMAN, A., DE OLIVEIRA, D., MATTOSO, M., 2023, "Deep Learning Provenance Data Integration: A Practical Approach". In: *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, p. 1542–1550, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450394192. doi: 10.1145/3543873.3587561. Disponível em: <https://doi.org/10.1145/3543873.3587561>.

PINA, D., CHAPMAN, A., KUNSTMANN, L., DE OLIVEIRA, D., MATTOSO, M., 2024, "DLProv: A Data-Centric Support for Deep Learning Workflow Analyses". In: *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*, DEEM '24, p. 77–85, New York, NY, USA. Association for Computing Machinery. ISBN: 9798400706110. doi:

10.1145/3650203.3663337. Disponível em: <https://doi.org/10.1145/3650203.3663337>.

PINA, D., KUNSTMANN, L., BEVILAQUA, F., SIQUEIRA, I., LYRA, A., DE OLIVEIRA, D., MATTOSO, M., 2022, "Capturing Provenance from Deep Learning Applications Using Keras-Prov and Colab: a Practical Approach", *Journal of Information and Data Management*, v. 13, n. 5 (Dec.). doi: 10.5753/jidm.2022.2544. Disponível em: <https://sol.sbc.org.br/journals/index.php/jidm/article/view/2544>.

PINA, D., KUNSTMANN, L., DE OLIVEIRA, D., MATTOSO, M., 2025, "Breadcrumbs for your Deep Learning Model: Following Provenance Traces with DLProv", *Software Impacts*, v. 23, pp. 100730. ISSN: 2665-9638. doi: https://doi.org/10.1016/j.simpa.2024.100730. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2665963824001180>.

POLYZOTIS, N., ROY, S., WHANG, S. E., ZINKEVICH, M., 2018, "Data Lifecycle Challenges in Production Machine Learning: A Survey", *SIGMOD Rec.*, v. 47, n. 2 (dec), pp. 17–28. ISSN: 0163-5808. doi: 10.1145/3299887.3299891. Disponível em: <https://doi.org/10.1145/3299887.3299891>.

PORTELLA, G., NAKANO, E. Y., RODRIGUES, G. N., MELO, A. C. M. A., 2019, "Utility-Based Strategy for Balanced Cost and Availability at the Cloud Spot Market". In: Bertino, E., Chang, C. K., Chen, P., Damiani, E., Goul, M., Oyama, K. (Eds.), *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*, pp. 214–218. IEEE. doi: 10.1109/CLOUD.2019.00045. Disponível em: <https://doi.org/10.1109/CLOUD.2019.00045>.

PRUYNE, J., WOZNIAK, J. M., FOSTER, I., 2022, "Tracking Dubious Data: Protecting Scientific Workflows from Invalidated Experiments". In: *2022 IEEE 18th International Conference on e-Science (e-Science)*, pp. 456–461. doi: 10.1109/eScience55777.2022.00082.

RAISSI, M., PERDIKARIS, P., KARNIADAKIS, G. E., 2019, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", *Journal of Computational physics*, v. 378, pp. 686–707.

ROGERS, J., CRISAN, A., 2023, "Tracing and Visualizing Human-ML/AI Collaborative Processes through Artifacts of Data Work". In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450394215. doi: 10.1145/3544548.3580819. Disponível em: <https://doi.org/10.1145/3544548.3580819>.

ROSENDO, D., MATTOSO, M., COSTAN, A., SOUZA, R., PINA, D., VALDURIEZ, P., ANTONIU, G., 2023, "ProvLight: Efficient Workflow Provenance Capture on the Edge-to-Cloud Continuum". In: *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 221–233. doi: 10.1109/CLUSTER52292.2023.00026.

RUPPRECHT, L., DAVIS, J. C., ARNOLD, C., GUR, Y., BHAGWAT, D., 2020, "Improving reproducibility of data science pipelines through transparent provenance capture", *Proc. VLDB Endow.*, v. 13, n. 12 (ago.), pp. 3354–3368. ISSN: 2150-8097. doi: 10.14778/3415478.3415556. Disponível em: <https://doi.org/10.14778/3415478.3415556>.

SÁENZ-ADÁN, C., MOREAU, L., PÉREZ, B., MILES, S., GARCÍA-IZQUIERDO, F. J., 2018, "Automating provenance capture in software engineering with UML2PROV". In: *International Provenance and Annotation Workshop*, pp. 58–70. Springer.

SCHELTER, S., BIESSMANN, F., JANUSCHOWSKI, T., SALINAS, D., SEUFERT, S., SZARVAS, G., 2015, "On challenges in machine learning model management", *IEEE Data Engineering Bulletin*. Disponível em: <https://www.amazon.science/publications/on-challenges-in-machine-learning-model-management>.

SCHELTER, S., BÖSE, J.-H., KIRSCHNICK, J., KLEIN, T., SEUFERT, S., 2017, "Automatically tracking metadata and provenance of machine learning experiments". In: *Machine Learning Systems workshop at the conference on Neural Information Processing Systems (NIPS)*.

SCHERZINGER, S., SEIFERT, C., WIESE, L., 2019, "The Best of Both Worlds: Challenges in Linking Provenance and Explainability in Distributed Machine Learning". In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1620–1629. IEEE.

SCHLEGEL, M., SATTLER, K.-U., 2023a, "Management of Machine Learning Lifecycle Artifacts: A Survey", *SIGMOD Rec.*, v. 51, n. 4 (jan), pp. 18–35. ISSN:

0163-5808. doi: 10.1145/3582302.3582306. Disponível em: <https://doi.org/10.1145/3582302.3582306>.

SCHLEGEL, M., SATTLER, K.-U., 2023b, "MLflow2PROV: Extracting Provenance from Machine Learning Experiments". In: *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*, DEEM '23, New York, NY, USA, b. Association for Computing Machinery. ISBN: 9798400702044. doi: 10.1145/3595360.3595859. Disponível em: <https://doi.org/10.1145/3595360.3595859>.

SCHLEGEL, M., SATTLER, K.-U., 2024, "Capturing end-to-end provenance for machine learning pipelines", *Information Systems*, p. 102495. ISSN: 0306-4379. doi: https://doi.org/10.1016/j.is.2024.102495. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437924001534>.

SCHMIDHUBER, J., 2015, "Deep learning in neural networks: An overview", *Neural Networks*, v. 61, pp. 85–117. ISSN: 0893-6080. doi: https://doi.org/10.1016/j.neunet.2014.09.003. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608014002135>.

SERGEEV, A., BALSO, M. D., 2018. "Horovod: fast and easy distributed deep learning in TensorFlow". Disponível em: <https://arxiv.org/abs/1802.05799>.

SHALEV-SHWARTZ, S., BEN-DAVID, S., 2014, *Understanding Machine Learning: From Theory to Algorithms*. USA, Cambridge University Press. ISBN: 1107057132.

SHANKAR, S., PARAMESWARAN, A. G., 2022, "Towards Observability for Production Machine Learning Pipelines", *Proc. VLDB Endow.*, v. 15, n. 13 (sep), pp. 4015–4022. ISSN: 2150-8097. doi: 10.14778/3565838.3565853.

SILVA, F., PINA, D., KUNSTMANN, L., MATTOSO, M., 2021a, "Painel de Proveniência: análise durante o treinamento de Redes Neurais Profundas". In: *Anais Estendidos do XXXVI Simpósio Brasileiro de Bancos de Dados*, pp. 22–28, Porto Alegre, RS, Brasil, a. SBC. doi: 10.5753/sbbd_estendido.2021.18158. Disponível em: <https://sol.sbc.org.br/index.php/sbbd_estendido/article/view/18158>.

SILVA, R., PINA, D., KUNSTMANN, L., DE OLIVEIRA, D., VALDURIEZ, P., COUTINHO, A., MATTOSO, M., 2021b, "Capturing Provenance to Improve the Model Training of PINNs: first handon experiences with Grid5000". In:

42nd Ibero-Latin-American Congress on Computational Methods in Engineering and 3rd Pan American Congress on Computational Mechanics, pp. 1–7, b.

SILVA, V., CAMATA, J., DE OLIVEIRA, D., COUTINHO, A. L. G. A., VALDURIEZ, P., MATTOSO, M., 2016. "In Situ Data Steering on Sedimentation Simulation with Provenance Data". SC: High Performance Computing, Networking, Storage and Analysis, nov. Disponível em: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-01400532>. Poster.

SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., MATTOSO, M., 2018, "DfAnalyzer: runtime dataflow analysis of scientific applications using provenance", *VLDB*, v. 11, pp. 2082–2085.

SILVA, V., CAMPOS, V., GUEDES, T., CAMATA, J., DE OLIVEIRA, D., COUTINHO, A. L., VALDURIEZ, P., MATTOSO, M., 2020, "DfAnalyzer: Runtime dataflow analysis tool for Computational Science and Engineering applications", *SoftwareX*, v. 12, pp. 100592.

SMITH, S. L., KINDERMANS, P.-J., LE, Q. V., 2018, "Don't Decay the Learning Rate, Increase the Batch Size". In: *International Conference on Learning Representations*. Disponível em: <https://openreview.net/forum?id=B1Yy1BxCZ>.

SOUZA, R., 2019, *Supporting User Steering In Large-Scale Workflows With Provenance Data*. Tese de Doutorado, UFRJ, Rio de Janeiro.

SOUZA, R., AZEVEDO, L. G., LOURENÇO, V., SOARES, E., THIAGO, R., BRANDÃO, R., CIVITARESE, D., VITAL BRAZIL, E., MORENO, M., VALDURIEZ, P., MATTOSO, M., CERQUEIRA, R., NETTO, M. A. S., 2022, "Workflow provenance in the lifecycle of scientific machine learning", *Concurrency and Computation: Practice and Experience*, v. 34, n. 14, pp. e6544. doi: https://doi.org/10.1002/cpe.6544. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6544>.

SOUZA, R., CAINO-LORES, S., COLETTI, M., SKLUZACEK, T. J., COSTAN, A., SUTER, F., MATTOSO, M., DA SILVA, R. F., 2024, "Workflow Provenance in the Computing Continuum for Responsible, Trustworthy, and Energy-Efficient AI". In: *2024 IEEE 20th International Conference on e-Science (e-Science)*, pp. 1–7. doi: 10.1109/e-Science62913.2024.10678731.

STILLER, P., BETHKE, F., BÖHME, M., PAUSCH, R., TORGE, S., DEBUS, A., VORBERGER, J., BUSSMANN, M., HOFFMANN, N., 2020, "Large-scale neural solvers for partial differential equations". In: *SMC 2020*, pp. 20–34. Springer.

TSAY, J., MUMMERT, T., BOBROFF, N., BRAZ, A., WESTERINK, P., HIRZEL, M., 2018, "Runway: machine learning model experiment management tool". In: *Conference on systems and machine learning (sysML)*.

VALENTINE, D., ZASLAVSKY, I., RICHARD, S., MEIER, O., HUDMAN, G., PEUCKER-EHRENBRINK, B., STOCKS, K., 2021, "EarthCube Data Discovery Studio: A gateway into geoscience data discovery and exploration with Jupyter notebooks", *Concurrency and computation: practice and experience*, v. 33, n. 19, pp. e6086.

VARTAK, M., MADDEN, S., 2018, "Modeldb: Opportunities and challenges in managing machine learning models." *IEEE Data Eng. Bull.*, v. 41, n. 4, pp. 16–25.

VARTAK, M., SUBRAMANYAM, H., LEE, W.-E., VISWANATHAN, S., HUSNOO, S., MADDEN, S., ZAHARIA, M., 2016, "ModelDB: a system for machine learning model management". In: *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pp. 1–3.

WANG, D., WEISZ, J. D., MULLER, M., RAM, P., GEYER, W., DUGAN, C., TAUSCZIK, Y., SAMULOWITZ, H., GRAY, A., 2019a, "Human-AI collaboration in data science: Exploring data scientists' perceptions of automated AI", *Proceedings of the ACM on Human-Computer Interaction*, v. 3, n. CSCW, pp. 1–24.

WANG, D., CHURCHILL, E., MAES, P., FAN, X., SHNEIDERMAN, B., SHI, Y., WANG, Q., 2020, "From Human-Human Collaboration to Human-AI Collaboration: Designing AI Systems That Can Work Together with People". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–6.

WANG, Q., MING, Y., JIN, Z., SHEN, Q., LIU, D., SMITH, M. J., VEERAMACHANENI, K., QU, H., 2019b, "ATMSeer: Increasing Transparency and Controllability in Automated Machine Learning". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, p. 1–12, New York, NY, USA, b. Association for Computing Machinery. ISBN: 9781450359702. doi: 10.1145/3290605.3300911. Disponível em: <https://doi.org/10.1145/3290605.3300911>.

WOZNIAK, J. M., CHARD, R., CHARD, K., NICOLAE, B., FOSTER, I., 2022a, "XD/ML Pipelines: Challenges in Automated Experimental Science Data Processing". In: *ASCR Workshop on the Management and Storage of Scientific Data*, a.

WOZNIAK, J. M., LIU, Z., VESCOVI, R., CHARD, R., NICOLAE, B., FOSTER, I., 2022b, "Braid-DB: Toward AI-Driven Science with Machine Learning Provenance". In: Nichols, J., Maccabe, A. B., Nutaro, J., Pophale, S., Devineni, P., Ahearn, T., Verastegui, B. (Eds.), *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*, pp. 247–261, Cham, b. Springer International Publishing. ISBN: 978-3-030-96498-6.

XIN, D., MA, L., LIU, J., MACKE, S., SONG, S., PARAMESWARAN, A., 2018, "Accelerating Human-in-the-Loop Machine Learning: Challenges and Opportunities". In: *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450358286. doi: 10.1145/3209889.3209897. Disponível em: <https://doi.org/10.1145/3209889.3209897>.

XIN, D., WU, E. Y., LEE, D. J.-L., SALEHI, N., PARAMESWARAN, A., 2021, "Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450380966. doi: 10.1145/3411764.3445306. Disponível em: <https://doi.org/10.1145/3411764.3445306>.

YAO, C., AGRAWAL, D., CHEN, G., OOI, B. C., WU, S., 2016, "Adaptive Logging: Optimizing Logging and Recovery Costs in Distributed In-memory Databases". In: *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, p. 1119–1134, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450335317. doi: 10.1145/2882903.2915208. Disponível em: <https://doi.org/10.1145/2882903.2915208>.

YASUTAKE, B., SIMONSON, N., WOODRING, J., DUNCAN, N., PFEFFER, W., ASUNCION, H., FUKUDA, M., SALATHE, E., 2015, "Supporting provenance in climate science research". In: *Proc. 7the International Conference*

on Information, Process, and Knowledge Management-eKnow 2015, pp. 22–27.

ZAHARIA, M., CHEN, A., DAVIDSON, A., GHODSI, A., HONG, S. A., KONWINSKI, A., MURCHING, S., NYKODYM, T., OGILVIE, P., PARKHE, M., XIE, F., ZUMAR, C., 2018, "Accelerating the machine learning lifecycle with MLflow." *IEEE Data Eng. Bull.*, v. 41, n. 4, pp. 39–45.

ZHU, Y., ZABARAS, N., 2018, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification", *Journal of Computational Physics*, v. 366, pp. 415–447.