Thesis for The Degree of Doctor of Philosophy

Towards Next-Gen Machine Learning Asset Management Tools

Idowu O. Samuel

UNIVERSITY OF GOTHENBURG

Division of Interaction Design and Software Engineering
Department of Computer Science & Engineering
University of Gothenburg
Gothenburg, Sweden, 2023

Towards Next-Gen Machine Learning Asset Management Tools

Idowu O. Samuel

Department of Computer Science & Engineering
Division of Interaction Design and Software Engineering
University of Gothenburg
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2023.

“To know, is to know that you know nothing. That is the meaning of true knowledge.”

- Socrates

Abstract

Context: The proficiency of *machine learning* (ML) systems in solving many real-world problems effectively has enabled a paradigm shift toward ML-enabled systems. In ML-enabled software, significant software code artifacts (i.e., assets) are replaced by ML-related assets, introducing multiple system development and production challenges. In particular, the need to manage extended asset types introduced by ML systems and the non-deterministic nature of ML make using traditional *software engineering* (SE) tools ineffective. The lack of supporting tools makes it demanding to address the concerns of specific aspects of ML-enabled system development, such as model experimentation. Consequently, new tool classes are being introduced to address these challenges. ML *experiment management tools* (ExMT) are examples of such tools aiming to mitigate the challenges and users' burden of managing ML-specific assets. Although these tools have recently become available, they are, unfortunately, not fully mature and have the potential for several improvements. For instance, many practitioners still consider ExMTs costly, restrictive, and ineffective. These challenges imply the need for improvements in many areas and raise research questions about the appropriate characteristics of a useful and effective ExMT for managing the development assets of ML-enabled systems.
Objective: This PhD research aims to contribute to the rapidly evolving space of new and improved ExMTs to facilitate the development of improved tools targeting combined SE and data science use cases. Consequently, we contributed to the knowledge and extended insights on ML experiment, their assets, the ExMT's landscape, and their benefits and effectiveness. We later proposed steps towards integrated ExMTs and artifacts based on the obtained insights.
Method: We addressed our objectives by adopting 1) *knowledge-seeking* research, including exploratory studies, literature reviews, feature surveys, practitioner surveys, and controlled experiments, and 2) *solution-seeking* research, including design science proposing unified concepts from multiple tools. The former was used to understand ML experiments, the challenges of managing experiment assets, the state of practice and landscape of existing ExMTs, and their effectiveness, benefits, and limitations. The acquired insights are then leveraged to propose research steps in the later part toward integrated ExMTs using design science to develop a blueprint for unified management tools.
Results: This thesis presents seven significant results. First, it provides an empirically informed overview of the challenges in ML experiment management. Second, it presents insights into the types of ML-based projects, their development activities, and evolution patterns. Third, it offers an overview of existing tools, shedding light on the state of practice and research on asset management tools for ML experiments. Fourth, it presents an empirical-based report on the benefits and challenges of ExMTs. Fifth, it establishes the effectiveness of ExMTs in improving user performance. Sixth, it proposes a step-by-step guide toward integrated ML tools for SE and data science. Seventh, it presents a prototype and blueprint for a unified ExMT.
Conclusion: This thesis highlights the significance of ML asset management as an essential discipline in facilitating experiments and asset management for ML-enabled software systems. It provides empirical data that offers crucial insights into the tooling landscape for managing ML experiment assets, including their features, benefits, limitations, and effectiveness. Additionally, the research proposes a guide and prototype to facilitate the design of new ExMTs.

Keywords

Machine Learning, Workflow, Experiment, Management, Model Life-cycle, Asset, Artifact, Management, Tools, Systems, Taxonomy, Assessment, SE4ML

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Thorsten Berger, for providing me with the opportunity to pursue my Ph.D. studies and for his continuous support throughout my research journey. His unwavering guidance, patience, motivation, enthusiasm, and vast knowledge have been invaluable to me. I am truly grateful for his mentorship and expertise.

I would also like to extend my heartfelt thanks to my co-supervisor, Daniel Strüber, for his invaluable support, guidance, and encouragement. His expertise and insights have greatly contributed to the success of my research and the completion of this thesis. I am fortunate to have had such a dedicated and supportive co-supervisor.

I am grateful to my supervisors for my Licentiate studies and research, Christer Åhlund, Saguna Saguna, and Olov Schélen of the Pervasive Computing research group at Luleå University of Technology, Sweden.

I would like to acknowledge the EASELab Group and my fellow researchers for their stimulating discussions, insightful comments, collaboration, encouragement, and guidance. I express my gratitude to Mukelabai Mukelabai, Dragule Swaib, Razan Ghzouli, Wardah Mahmood, Ricardo Caldas, Jacob Kruger, and Yorick Sens for their valuable contributions. I also extend my thanks to my colleagues Weixing Zhang, Afonso Fontes, and Peter Hazem Samoaa for their camaraderie and engaging conversations.

I am indebted to my managers at Aptiv for their support in various capacities and for granting me study leave to complete my Ph.D. studies.

To my caring, loving, and supportive wife, Mobolaji, I offer my deepest and sincerest gratitude. Your unwavering encouragement, prayers, and sacrifices during challenging times are immensely appreciated and cherished. I would also like to express my love and gratitude to my adorable boys, Fola and Folu, who have been a constant source of inspiration for me.

To my parents and siblings, I extend my heartfelt thanks for their unwavering belief in me and their continuous encouragement to strive for excellence.

Lastly, I would like to thank God for granting me the grace and strength to complete this thesis. I extend my gratitude to all those who have supported me throughout this journey in various ways—whether by answering my questions, offering encouragement, or simply being good friends and colleagues.

Samuel, O. Idowu

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **S. Idowu**, C. Åhlund, and O. Schelén "Machine Learning in District Heating System Energy Optimization"
in PerCom Workshops, pp. 224–227, 2014.

[B] **S. Idowu**, S. Saguna, C. Ahlund, and O. Schelen "Applied Machine Learning: Forecasting Heat load in District Heating System"
Elsevier Energy and Buildings, vol. 133, pp. 478–488, 2016.

[C] **S. Idowu**, O. Osman, D. Struber and T. Berger "On the Effectiveness of Machine Learning Experiment Management Tools"
IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (SEIP), pp. 207-208, 2022.

[D] **S. Idowu**, D. Strüber, and T. Berger "Asset Management in Machine Learning: State-of-research and State-of-practice."
ACM Computing Surveys (CSUR), 55(7): 144:1-144:35, 2022.

[E] **S. Idowu**, O. Osman, D. Struber, and T. Berger "Machine Learning Experiment Management Tools: A Mixed-Method Empirical Study"
Empirical Software Engineering (EMSE), 2023, [Under minor revision].

[F] **S. Idowu**, Y. Sens, T. Berger, M. Vierhauser, and J. Kruger "A Large-Scale Study of ML-Related Python Projects"
2024, [Under review].

[G] **S. Idowu**, D. Strüber, and T. Berger "EMMM: A Unified Meta-Model for Tracking Machine Learning Experiments"
Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2022.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] **S. Idowu**, S. Saguna, C. Åhlund, and O. Schelén "Forecasting heat load for smart district heating systems: A machine learning approach" *IEEE International Conference on Smart Grid Communications, pp. 554–559, 2014.*

[b] **S. Idowu**, D. Struber, and T. Berger "Asset Management in Machine Learning: A Survey" *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (SEIP), pp. 51–60, 2021.*

[c] M. Ahmed, K. Bassuday, **S. Idowu**, W. Leeson, D. Struber, T. Berger "Designing and Deploying Software Defect Prediction in Industry: Experiences from the Infotainment Domain" *[Unpublished].*

Research Contribution

I contributed to several papers, and my contributions are listed below according to the Contributor Roles Taxonomy (CRediT)[1]. The degree of my contribution is specified as 'lead,' 'equal,' or 'supporting' where necessary.

`Papers A` and `B` report exploratory studies on Machine Learning (ML) in the District Heating System (DHS) domain. I equally participated in conceptualizing the study, data curation, and formulating the methodology. However, I led the formal analysis and implementation of the study. I conducted extensive investigations on the subject systems by reviewing state-of-the-art literature on energy prediction methods and acquiring ML and DHS domain knowledge. I led the experimentations and visualization of the research results. I also led the writing process for the original draft, as well as the reviewing and editing of the papers.

`Paper C` proposes a research agenda towards integrated ML and software engineering tools. Although I equally participated in the conceptualization, I led the investigation of the subject area and the visualization of the research result. I took charge of the writing process for the original paper and contributed equally to the review and editing of the paper.

`Paper D` presents the report of a literature review and feature model analysis, which is an extension of `Paper b` with the same authors. While I equally participated in conceptualizing the original research paper, I led the data curation and collection of relevant resources such as subject literature and tools. I also led the formal analysis and investigation of the subject tools. I equally participated in the methodology design and led the implementation of the study. I also led the visualization and presentation of the result data. I took charge of the writing process for both `Paper D` and `Paper b` and led the review and editing of the papers.

`Paper E` reports two empirical studies. For the first empirical study using a controlled experiment, I led the conceptualization of the research idea and equally participated in designing the controlled experiment and questionnaire. I also equally participated in the methodology design and the data curation by disseminating the experiment information and instructions to participants. I led the analysis of participants' responses and the result presentation. I equally participated in conceptualizing the research idea for the practitioners' survey and led the survey questionnaire design. I also equally participated in the data collection by disseminating survey instruments to potential participants. I led the analysis of participants' responses and the result presentation. Moreover, I led the writing of the original drafts and had a hand in reviewing and editing the final paper.

`Paper F` reports on a large-scale study adopting repository mining on ML-related Python projects. Although I equally participated in conceptualizing the research idea, I supported the data curation and collection. I also equally participated in the methodology design and analysis of the subject projects, led the visualization of the result data, and contributed to writing the original drafts and final paper.

`Paper G` adopts design science to propose artifacts unifying multiple tools' versioning structures and concepts. Although I equally participated in the

[1]https://credit.niso.org/

conceptualization of the paper, I led the formal analysis and modeling of the resulting metamodel and led the validation of the metamodel. I took charge of the writing process for the original draft, review, and editing of the paper.

Table 1: Summary of personal contributions per paper according to the Contributor Roles Taxonomy (CRediT)

	Conceptualization	Data Curation	Formal Analysis	Funding Acquisition	Investigation	Methodology	Project Administration	Resources	Software	Supervision	Validation	Visualization	Writing - Original Draft	Writing - Review & Editing
Paper A	✓		✓		✓	✓					✓	✓	✓	✓
Paper B	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓
Paper C	✓		✓		✓								✓	✓
Paper D	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓
Paper E	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓
Paper F	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓
Paper G	✓	✓	✓		✓	✓	✓		✓		✓	✓	✓	✓

Contents

Chapter 1

Introduction

Machine Learning (ML) has proven effective in solving many present and future issues. To take advantage of its benefits, ML components must be integrated into new and existing software systems. For instance, modern email clients are now equipped with intelligent capabilities to classify emails by type and urgency and generate new emails based on the target audience and context. As a result, an increasing number of ML-enabled software systems are being developed, signaling a shift towards ML-enabled software systems [1–3]. This shift is evident in the willingness of companies to embrace ML. More and more businesses are now eager to incorporate ML into their software products and services, transforming themselves into companies that utilize ML as a natural component [1, 4]. We refer to these software systems with ML capability as *ML-enabled systems* (also known as ML-based software systems or Software 2.0 systems).

There are various implications of the emergence of ML-enabled software systems on software engineering (SE) and development [5–10]. One of such noteworthy challenges is the complexity associated with the collaboration between two different disciplines—SE and Data Science—with differing cultures and mindsets regarding goals and focus. For example, software engineers are typically concerned about system performance, costs, stability, release time, and balancing trade-offs. However, data scientists often work on a fixed dataset for training and evaluation, with an experimentation and prototyping perspective. They tend to focus more on model accuracy as primary performance and less on model size constraints, latency, or the ability to update models, which are significant considerations for software engineers. Also, while traditional software engineers think of requirements and specifications in terms of interfaces and contracts, which helps define systems' expectations, such a concept is missing in data science. In contrast, specifications are often observed as goals (e.g., maximizing a specific objective) since ML components are primarily black boxes [11].

In addition, ML's non-deterministic nature complicates ML systems' engineering [12, 13]. For example, testing ML software introduces several engineering complexities, making applying traditional software methods directly impractical.

Various complications and challenges resulting from the emergence of ML-enabled systems discussed in the literature can be grouped into three areas—Development, Production, and Organizational [4]. Development challenges affect the engineering practices when building ML systems and components and often apply from the system requirement stage to deployment. Production challenges affect the practical operation of ML-based systems during and after deployment. Some issues that arise during the production stages are variability in the data affecting ML system reliability, improper pre-processing of the data leading to prediction inaccuracy, and algorithmic bias due to data drifting.

The development and production-related challenges have led to a rapidly growing research interest in addressing challenges to improve the engineering practices of ML-based systems. Although early reports on the challenges of engineering ML systems were commonly reported as lessons from industrial practices [14, 15], growing research communities under the term SE4ML and AI Engineering [16–18] are interested in exploring the different facets of SE transformation with its paradigm shift. The interest of these research communities includes interdisciplinary collaboration between SE and data science disciplines, quality assurance, system-level thinking, safety, and improved or new development tools. They seek to answer relevant questions such as *How do we effectively build and maintain systems that have an ML component in them? How do we integrate ML as a component in systems where such components are either core or a small part of an extended system?*

Due to the paradigm shift toward ML-enabled systems, it becomes essential to improve existing tools and develop new ones accommodating their uniqueness. While traditional SE tools have evolved over several years to handle traditional software artifacts (i.e., assets) effectively, studies have established that they are inefficient for broader asset types required by ML systems [19–21]. Such studies have highlighted the challenges of applying traditional SE methods and tools to facilitate the creation of ML-enabled systems [4, 8, 22–24]. Notably, they have reported using traditional SE tools, such as version control systems, in the ML component context as ineffective for different reasons. Hence, a need for new and dedicated methods and tools [9, 19–21, 25].

There are several reasons why it is important to have new and improved tooling support for developing ML-based software systems. Firstly, the variety of asset types has increased, which means that tools must provide support and operations on the appropriate level of abstraction. Additionally, ML component development is often experimental due to the non-deterministic nature of these systems. Furthermore, effectively addressing development concerns unique to ML systems, such as reproducibility, is difficult. Lastly, with practitioners increasingly performing both SE and data science roles, there is a need for better tooling support that meets existing SE standards to manage ML assets effectively.

To address these tooling gaps, new asset management tools have emerged, targeting the development of ML-based systems. Most of these have been championed by the industrial need to address critical development concerns such as ML-specific asset versioning, provenance, and reproducibility [26–31]. For example, there has been a surge of various ML-specific asset management tools with key features such as workflow management, pipeline management, model management, dataset & feature management, and experiment management tools (ExMT) [32]. When comparing the granularity of their offered management support to traditional tools such as version control systems, we identify the ML experiment management features as fundamental to adequately address several development challenges of ML-enabled systems. Also, the explicit management of ML experiments and their associated assets is significant in reducing the complexity and time overhead of managing assets in multiple projects [19–21], hence facilitating ML-driven software development. Consequently, this thesis focuses on the scope of ML ExMTs as a subset of ML asset management.

Although new ExMTs have recently become available, they are not fully matured. ML asset management tools with experiment management features, such as MLFlow, Neptune, and DVC, offer practical ways to maintain an account of asset provenance during ML experiments to support different development concerns. However, as these tools have become available recently, they are potentially not fully grown, and practitioners face several challenges and limitations when using them. Mora-Cantallops et al. [31] highlight the lack of interoperability across different tools, lack of explicit representation of domain knowledge, and friction or overhead incurred during usage as factors hindering the adoption of these tools. Other limitations include a lack of flexibility or customization to meet varying use cases; limited integration support, especially for standard SE tools; limited collaboration, making it challenging to use in large-scale industry settings; lack of robustness and constant availability.

New and improved tools should address these limitations in existing tools by incorporating the domain knowledge about ML experiments obtained from investigating the existing tool landscape. Also, there is a strong potential for integrating ML experiment management support into existing traditional software tools. For example, new and improved version control tools, building on traditional ones and extending them with domain-specific operations tailored to ML assets. Such tools would address ML-specific challenges from the perspective of software engineers, who routinely use standard traditional tools such as Git.

The objective of this thesis is to enhance the development of next-generation tools that can support both traditional software assets and ML. The tools should also integrate seamlessly with existing tools like Integrated Development Environments (IDEs) and Version Control Systems (VCS) and cater to various development use cases and modalities. To achieve this goal, we propose using research techniques like systematic literature, features and practitioner surveys, controlled experiments, exploratory studies, and design science. Specifically, the concrete contributions of this thesis are:

C1: Empirically informed overview of the challenges of managing ML experiment assets without specialized tools.

C2: Insights and empirical data on the types of ML-based projects, their development stages, and evolving patterns in ML experiment projects.

C3: Overview of existing ExMTs, shedding light on variability/commonalities and features of state-of-the-art tools.

C4: Empirical-based report on the benefits and limitations of ExMTs.

C5: Establishment of the effectiveness of ML ExMTs on user performance.

C6: Research guide toward integrating ML experiment management into traditional SE tools.

C7: Blueprint on how to unify and integrate multiple ML ExMTs.

In addition to these main contributions, this thesis provides an overarching contribution of positioning ML asset management as an essential discipline to facilitate experiment and asset management for ML-enabled software systems [33]. The overall thesis contributions will provide insights and inspire follow-up work to researchers and tool builders on assessing, improving, and developing new asset management tools for developing ML-based systems while increasing values for ML and software engineers.

In the following chapters, we discuss the background of ML development workflow, ML assets, and management tools in Section 1.1, followed by the thesis's research scope and goals in Section 1.2. We present the methodologies adopted to address our research goal in Section 1.3, followed by our contributions and their discussions in Section 1.4. We summarize the seven papers comprising this thesis in Section 1.5, and conclude with thesis conclusion in Section 1.6

1.1 Background

In this section, we will cover the fundamentals of developing ML components, including model prototyping through ML experiments, and tools for managing ML assets.

1.1.1 ML Development Workflow

The traditional SE process [34] includes requirements analysis, planning, architecture design, coding, testing, deployment, and maintenance stages. Similarly, supervised ML follows well-defined processes grounded in workflows designed in the data science and data mining context. Examples include CRISP-DM [35], KDD [36], and TDSP [37]. Figure 1.1 shows a simplified supervised ML development workflow structured along different development stages. The workflow consists of four primary stages: i) requirements analysis, ii) data-oriented works, iii) model-oriented works, and iv) DevOps works [4,8,9]. Figure 1.1 also shows that ML projects can either be production-focused or non-production-focused. For example, ML projects for research papers are often non-production focused and do not require DevOps operations. In contrast, ML-enabled software systems are production-focused because they usually integrate and operationalize models.

Requirement analysis stages

The requirements analysis stages are the first development steps involving the analysis of the system requirements and resources, such as data availability. The involved steps include the following:

System requirements. The ML component requirements are established at the start of ML-enabled software projects. While non-production often focuses on metrics such as prediction accuracies, production-focused systems

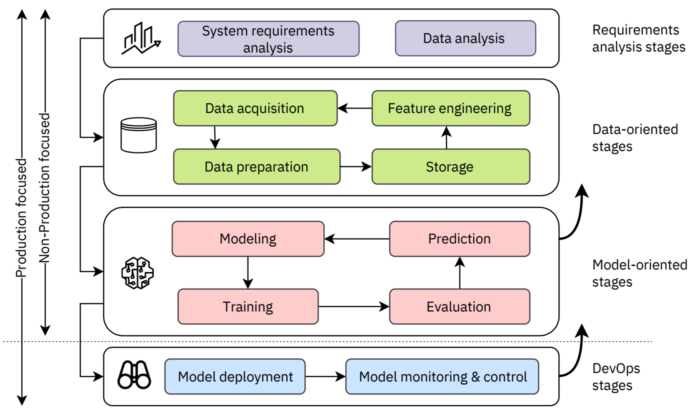

Figure 1.1: Development workflow stages of production-focused and non-production-focused ML components.

have extended requirements, such as learning type (e.g., online vs. offline) and model serving latency.

Data analysis. Initial exploratory data analyses are performed to derive useful information and insights that impact the project decision. For example, this may involve analyzing the application domain, data availability, sources, and appropriate domain and data features.

Data-oriented stages.

Based on the outcome of the system requirement stages, data-related activities to collect, pre-process, and prepare data for model training are carried out next. The involved steps include the following:

Data acquisition. Involves practical actions to collect raw data from identified sources. This stage often involves a vast amount of unstructured data to derive actionable insights. The primary focus in this stage includes data quality, completeness, consistency, and relevance. Example tasks under this stage include *Load*, *Collect*, *Obtain*, *Capture*, and *Survey*.

Data preparation. Data preparation is a tedious and time-consuming process, often requiring substantial resources, including human expertise and computational resources, to process the acquired raw data into presentable or usable structures. Some processes can be performed automatically using ETL (Extract-Transform-Load) tools. These tools perform different data processing steps, from data loading to formatting for analysis and output to data warehouse. Examples of tasks under this stage include *explore*, *wrangle*, *clean*, *filter*, *organize*, *standardize*, and *de-duplicate*.

Storage. ML development activities center around data processing; hence, for effective data processing, it is essential to find the right hardware/software combination for data persistence. For example, newly extracted features must be stored for further development activities such as modeling and training. Example tasks under this stage include *preserve*, *archive*, *log*, and *recycle*.

Feature engineering. Feature engineering involves selecting and extracting data features or variables appropriate for building specific ML models. It often requires the creation of new data features from the raw data to improve model performance towards specific requirements (e.g., accuracy or training/inference speed). Common tasks under this stage include *feature-select* or *construct*, *label*, and *annotate*.

Model-oriented stages.

These stages are essential to ML model development, where ML learning methods and algorithms are used to learn patterns from input pre-processed datasets. The involved steps include the following:

Modeling. The modeling stage includes model planning and selection and data mining to discover essential properties of data concerning specific ML tasks. This stage also involves selecting suitable data processing techniques and ideal ML algorithms. Common tasks under this stage include *classify*, *cluster*, *mine*, *analyze*, and *process*

Training. The modeling step is often followed by a training model using selected features and labeled data. The focus of this stage is on the optimization of model performance via multiple training iterations on hyperparameter search. Common tasks under this stage include *tune* and *optimize.*

Evaluation. Trained models are tested on various metrics, such as accuracy and latency performance. The testing stage should be performed on real-world data to assess the model's performance under production and non-production conditions. Several evaluation metrics include classification accuracy, confusion matrix, loss function, and error rates. Common tasks under this stage include *validate*, *test*, *verify*, and *review.*

Prediction. Prediction involves using a trained model on unseen examples (i.e., unlabeled data). The prediction capability of a model can also be evaluated by comparing its performance across validation, test, and training datasets.

DevOps stages.

The DevOps (or MLOps) stages include integrating and deploying ML models as software components with other ML or software components. These stages also involve the management of infrastructure and operations of ML components—monitoring and controlling—in production. The involved steps include the following:

Model deployment. ML model deployment is critical for production-focused projects, such as customer churn prediction or product recommendation. It concerns using an ML model that has met specific requirements in offline training to perform the same task on production data. This step also concerns effective ways to make the predictions robust and scalable for large datasets and effectively update models as required.

Model monitoring & control. Beyond deployment methods, monitoring and controlling models' behavior is essential to ensure that in-production models produce accurate and actionable results over time. This is particularly important if the model is used in production environments with large volumes of data where high predictive accuracy and reliability are essential to ensure business value.

1.1.2 ML Experiments & Model Prototyping

In practice, much development time goes into establishing viable ML models through ML experiments and model prototyping [20, 21]. These are often performed ahead of establishing a production-ready development pipeline. In some settings, multiple practitioners experiment on data features provided through feature stores to obtain the best-performing models. *ML experiments* are multiple incremental iterations performed over the development workflow stages as experiment *runs* or *trials*. The required exploratory and experimentation approaches to ML experiment and model prototyping are the primary factors in the different development nature of ML-enabled systems to traditional SE ones. As shown in Fig. 1.1, the ML workflow contains a linear progression from requirements analysis to DevOps stages; however, ML workflows are typically non-linear and include multiple feedback loops (indicated by the upward arrows) [9]. These feedback loops reflect the multiple experiment *runs*. We describe a *run* as a one-time cycle through the relevant workflow stages, often resulting in a trained model. Each run employs specific assets' versions (e.g., datasets, hyperparameters, source code) within the solution space of a particular ML task. The solution space includes required assets such as datasets from the application domain presenting relevant features for the learning task, a slice or subset of the initial dataset as training data, learning algorithms, and their (hyper)-parameters. A completed run's outcome often includes a trained model, the model performance measurement based on test data, and obtained predictions from an unseen slice or subset of the initial dataset. To find well-performing models, practitioners rely on multiple instances of trial-and-error steps due to the unpredictable nature of ML model performance [9, 38, 39].

Consequently, experiment runs are repeatedly performed while modifying or using new assets until the process results in a model that meets a specific target objective [4]. Such modification includes adding, removing, or engineering features, changing learning algorithms, testing different hyperparameters, and using various performance evaluation metrics. The decision to perform new runs is usually based on the result analysis of a current run and its model. Also, during the DevOps works (deployment, monitoring, and control of models), there is often a need to modify and make new experiment runs based on newly available data or drift corrections to ensure models stay within the target objective's course.

Figure 1.2 illustrates different asset types modified within a specific run's solution space. Model training involves training datasets, features, learning algorithms, and hyperparameters. In contrast, the model evaluation involves test datasets, models, predictions, and performance measures. The need to carry out multiple runs is often based on the analysis Y_x of model requirements and resulting model performance M_x when tested with dataset E_x; however, a user may use other requirement metrics to decide if a new run is required.

A manual or automatic approach may be employed to find the best-performing combinations of the asset versions over several runs. The manual approach follows experts' decisions on necessary step-by-step modifications within the solution space for new runs. In contrast, the automatic approach—AutoML [40–43]— systematically searches a pre-defined portion of the solution space (e.g., a set of hyper-parameters range) for each run. For

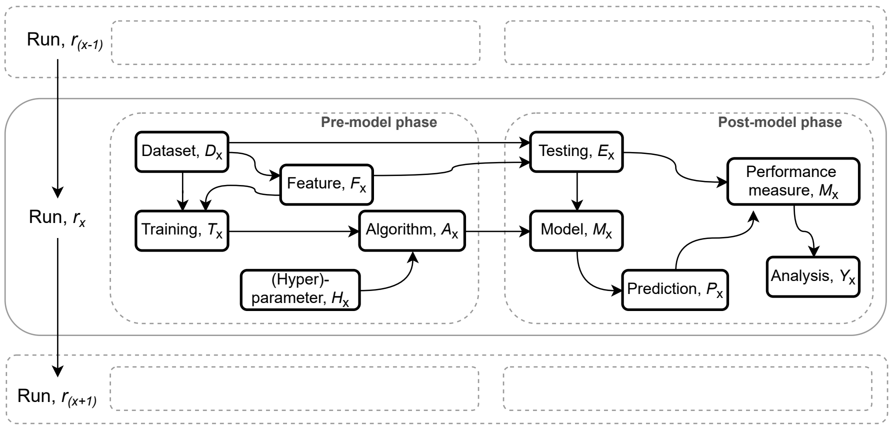

Figure 1.2: A representation of an ML experiment run

example, ML models can be automatically selected and parameterized through training loops. Regardless of the employed approach, several experiment runs are often performed before finding optimal models. The need for asset management support is often attributed to the complexity and time overhead that arises with manually managing the large number of asset versions resulting from the multiple exploratory runs [19–21].

1.1.3 ML Asset Management

Conventionally, the term *asset* is used for an item that has been designed for use in multiple contexts [44], such as a design, a specification, source code, a piece of documentation, or a test suite. ML practitioners often use the term *artifact* to describe different required resources during model development. Because of their experimental nature, these artifacts qualify as assets in the ML development context, which requires keeping artifacts for future use. Conventional SE primarily deals with source code artifacts and, therefore, often has fewer asset types compared to the engineering of ML-enabled systems. In contrast, ML includes additional artifact types, such as datasets, models, hyperparameters, and model evaluation metrics [45]. Consequently, we describe ML *assets* as individually storable units serving specific purposes in the development workflow of ML components.

When following the ML development workflow, a variety of assets are created, utilized, and modified. In the Data-oriented stages, datasets and features are used as input assets, and new feature sets, labels, or annotations are created. This process often requires software assets such as source code and environment dependencies and generates outputs like execution logs. During the Model-oriented stages, the outputs of the Data-oriented stages are used to develop and train models, and additional assets such as hyperparameters, metrics, model training-related source code, logs, and environment dependencies are generated or used. In the DevOps-related stages, trained ML models, their metadata, training execution logs, and dependencies (like libraries and environment runtimes) are utilized. Other assets created or used in these stages include software for model deployment and monitoring, such as wrapping trained models into web services API endpoints for on-demand or batched execution. During the operation of ML components in production, assets monitoring the inference execution and model performance (like metrics, statistics, and execution logs) are generated.

Consequent to the multiple versions of assets created, modified, and utilized during typical ML workflow, it is essential to employ efficient asset management measures to address development concerns such as traceability, reproduction, and collaboration throughout the development workflow stages. In the current state of practice, it is tempting to adopt traditional SE techniques such as VCS (e.g., Git) to address some of the highlighted asset management challenges. However, such tools were not designed to manage ML-specific assets nor support the intuitive and exploratory development approach of developing ML components. Consequently, to address the asset management challenges, there is a need for explicit management tools and methods that offer systematic ways to track, collect, organize, and manage assets used during model development and post-model creation.

In this light, we define *asset management* as an essential discipline to facilitate the engineering of ML-enabled systems:

Definition 1 (Asset Management)**.** The discipline of *asset management* comprises methods and tools for managing *ML assets* to facilitate activities involved in the development, deployment, and operation of ML-enabled systems. It offers *structures* for tracking and storing detailed ML assets of different types, as well as *operations* that practitioners can use to address practical management concerns.

This definition emphasizes that establishing effective asset management requires efficient storage and tracking structures (e.g., data schemas, types, modular and composable units, and interfaces) as well as properly defined operations, which can be of different modalities (e.g., command-line tools or APIs allowing IDE integration). Asset management extends to activities in practice and management areas covering different development aspects of ML-enabled systems.

1.1.4 ML Asset Management Tools

To provide adequate ML asset management, several tools and platforms support the systematic tracking, collection, storage, and management of ML assets over the various development stages of ML components, their deployment, and integration into software systems. We collectively refer to such systems as *ML asset management tools* (AMTs). The increasing popularity of such systems implies the growing need for effective asset management for ML-enabled software systems. Several categories currently exist to support the development activities of ML-enabled systems. Following previous work [32, 46], we classify them into five non-exclusive categories based on their functionality: i) Workflow management, ii) Pipeline management, iii) Model management, iv) Dataset and feature management, and v) Experiment management.

The non-exclusivity of the group indicates overlapping asset management functionalities across the different categories. For instance, tools such as

MLFlow can be described as model management tools or ExMTs.

The following paragraphs briefly explain these categories and discuss their asset management capabilities.

ML Workflow management.

This category focuses on the complete ML development workflow and provides management capabilities for all possible asset types under all development workflow stages. Such tools are often available as cloud-based platforms or standalone software tools. The cloud-based platform offers complete ML as a service (MLaaS) platforms. Examples include Microsoft Azure ML [47,48], Amazon SageMaker [48,49], Google Vertex AI [48,50], and DataRobot AI Platform [51]. This class of tools provides numerous ML services through the public cloud infrastructure, offering computing and memory resources subject to payments. The other class of workflow management tools comes as standalone software tools (or hybrid) that can be deployed on-premise. Examples of such tools include MLflow [52], Polyaxon [53], DVC [54] and Hopsworks [55]. A key aspect of workflow management tools is that they often offer management functionalities found in other management tool classes. For instance, MLFlow offers MLflow tracking, which essentially offers experiment management functionalities (described later). At the same time, it also provides an MLflow Model registry (model management), a specific tool for storing models, their metadata, dependencies, and lineage. MLFlow also offers MLFlow projects a standard format for packaging reusable development assets. DVC is another popular asset management tool that offers broad asset management functionality across its ecosystem. DVC asset versioning capability is built on traditional Git VCS, offering similar commands that effectively version ML-specific assets. Recently, the DVC ecosystem has grown to include the following tools: DVC Studio, an ExMT for tracking and sharing experiment assets; CML, an open-source CI/CD tool for ML projects; MLEM, an open-source model registry and deployment tool.

Pipeline management.

ML pipelines are abstractions that offer comprehensive views of the development stages of ML, including data processing, model development, and model deployment processes. The concrete presentations of ML pipelines are often implemented as direct acyclic graphs (DAGs) following ML workflow and its stages. Consequently, this class of management tools supports ML asset management through the defined pipelines and their related assets. Some workflow management tools offer pipeline-based management options (e.g., DVC); however, the pipeline management tools themselves typically lack support for DevOps-oriented stages of ML workflow. Examples of tools under this category providing similar pipeline-based asset management for different use cases include Velox [56], Vasma [57], and ArangoML [58]. Velox offers management of model training pipelines for predeclared models and model performance evaluation. Vasma offers support to automatically track the provenance of ML pipelines using static analyses of Python scripts, while ArangoML offers pipeline management to track lineage, audit, reproduce, and monitor models. Tools under this management class also include Apache SystemDS [59], MLtrace [60], ProvDB [61]. The Apache SystemDS is a declarative ML pipeline system offering abstraction for different ML workflow tasks. MLtrace and ProvDB offer support for tracking the lineage and provenance of ML pipeline assets.

Model management.

This class treats models and their metadata as central abstractions while providing focused support for managing the lifecycle of trained ML models. Their support is often limited to the Model- and DevOp-oriented stages of the ML workflow, with less support for data-related assets. Tools under this class have close or intersecting functionalities with the tools under the experiment management class (described below). In general, this class of tools offers efficient storage and retrieval of models, model selection, model comparison, monitoring, and model serving. They provide information on the lineage of model-related assets and various evaluation performances of models. Examples of tools under this class include ModelDB [20], and ModelHub [62], which focus on supporting model development, deployment, and monitoring. ModelDB stores and versions ML assets using a relational database, while ModelHub extends Git as an ML-specific version control system. ModelHub offers management CLI commands with native support for ML assets and unique ML development use cases. For example, discovering asset versions based on model qualities.

Tools focusing on efficient model storage include ModelKB [63], MMP, [64], and MISTIQUE [65]. ModelKB uses custom callbacks in native ML frameworks to collect metadata about ML experiments and automatically generate source code for deployment, sharing, and reproducibility. MMP is a unique model management platform for Industry 4.0 systems that integrates the business domain with the model metadata. MISTIQUE accelerates model training, evaluation, and interpretability by efficiently storing and managing model intermediates (e.g., hidden representation in deep neural networks) targeting big data and large-sized deep learning models.

Tools focusing on model serving include Clipper [66], BentoML [67] and TensorFlow-serving [68], and Cortex [69]. They offer ways to turn trained models into production API endpoints with few lines of code. While tools such as TensorFlow-serving only support TensorFlow-based models, other tools such as BentoML, Clipper, and Cortex offer multi-framework (Tensorflow, PyTorch, Scikit-Learn, XGBoost, FastAI, and more) support. Tools such as BentoML supports practitioners to deploy and test model in a few steps and provide operations support through scalable microservices.

Data-set & feature management.

The quality of datasets used in an ML model development plays a crucial role in the quality of generated models. Therefore, data-related development activities are crucial aspects of ML engineering. Tools under this class focus on the data-oriented stages of ML workflow. They often offer dataset, label, and feature storage and management functionalities with interfaces for data pre-processing, feature selection, and feature engineering. While their functionalities are often not found in other classes, they are often complementary tools in other management classes. Examples of tools in this class include MLdp [70], ExDra [71], and Pachyderm [72]. MLdp is a purpose-built data asset management tool that offers a flexible data model integrating various data types and supports large-volume data storage with data versioning, provenance, and integration with ML development frameworks. ExDra offers data acquisition, integration, and pre-processing support from federated and diverse raw data sources. In contrast, Pachyderm offers data-focused pipeline management with automated data versioning, containerized pipeline execution, and data provenance.

Experiment management.

Experiment management tools treat ML experiment runs as their central abstraction. Tools in this class aim to offer management support during the experimentation and model prototyping stage to reduce the cost, time, and complexities that burden manual or ad hoc asset management. Experiment management tools primarily offer reproducibility and post-experiment analysis for exploratory model development, training, and optimization. The operations offered by ExMTs complement model development frameworks (e.g., SciKitLearn and TensorFlow) and other management tool classes. ML ExMTs primarily offer functionalities to track asset states over multiple experiment *runs* in a structured and organized manner during model development workflow, focusing less on the Data-oriented and DevOps-oriented stages.

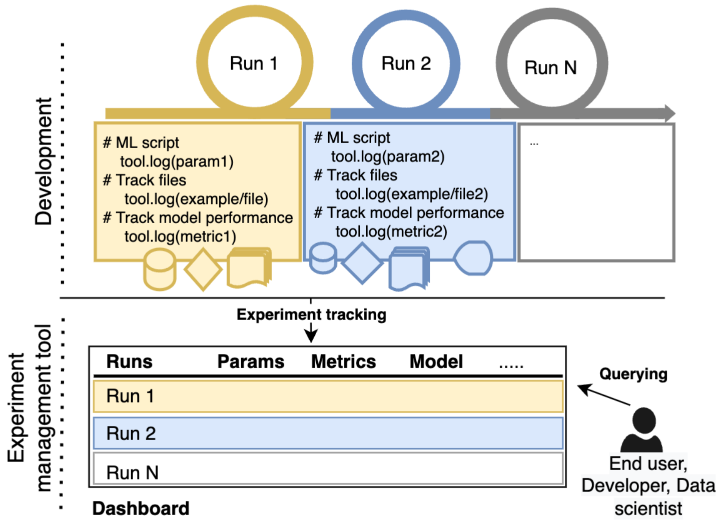

Figure 1.3: Illustration of how a typical API-based ExMT work

Figure 1.3 illustrates the workflow of ExMTs. For the user, they eliminate the risk of forgetting to track or commit important experiment milestones. Most tools capture and record a new version of modified assets when users execute an experiment run—indicating a complete run. For example, DVC offers the command *dvc exp run*, which executes a preconfigured experiment pipeline and simultaneously captures the versions of associated assets. Users can later access prior runs and their associated assets.

Examples of tools that fall under this class include MLFlow tracking [52], Neptune.ai [73], Studio ML [74], DVC, Sacred, and Weights & Biases (W&B)

[75], Comet [76]. MLflow tracking focuses on capturing, storing, and managing ML artifacts: MLflow Tracking is an API for logging experiment runs, including code and data dependencies, via automatic or manual instrumenting application code. These runs can be viewed, compared, and searched through an API or the Web dashboard UI. DVC For instance, with the help of its experiment management CLI commands, a snapshot of all supported assets is taken for each completed experiment run. StudioML supports non-intrusive asset collection, requiring no modification of existing experiment code. Other tools, such as Guild AI, Datmo, and Deepkit, provide experiment reproducibility-focused functionalities capturing all required assets (including source code, dependencies, execution environment, and logs) to rerun or reproduce an experiment.

Recent studies have presented taxonomies and surveys on ML ExMTs [77, 78]. These studies identify the *access modality* (the approach for tracking assets and how they can be queried for later use) as an essential paradigm of these tools. The tools' tracking approach is often *API-based* and/or *CLI-based*. API-based tools, such as Neptune.ai and MLFlow, collect assets intrusively, where users instruct the tools to track assets via provided APIs. In contrast, CLI-based tools, such as DVC and Datmo, require users to track experiment assets using the CLI. To query and retrieve stored assets for further analysis, the common methods offered by ExMTs are either GUI-based (dedicated web dashboard) or CLI-based paradigms.

It is important to note that the ML workflow, experiments, and asset management described also apply to Deep Learning, a type of ML that uses artificial neural networks [79].

1.2 Research Goal & Scope

This thesis aims to facilitate the development of asset management tools for ML-enabled software systems, specifically those in the experiment management category. We, therefore, carried out a number of *knowledge-seeking* research activities where we: i) Established the challenges faced when managing ML experiments without specialized tools. ii) Investigated development activities and evolution patterns in ML-related repositories for insight into ML experiments. iii) Investigated *state-of-practice* and *state-of-research* for ExMTs and the variabilities and commonalities of their features. iv) Performed a survey to elicit information on the benefits and limitations of ExMTs. v) Investigated the effectiveness of ExMTs on user performance. In addition to the knowledge-seeking research, we performed *solution-seeking* research where we: i) proposed a research guide on steps towards integrating existing ML ExMTs into traditional SE tools. ii) Proposed how researchers and tool developers can unify the structures and concepts of existing ML ExMTs into a common reusable artifact. Figure 1.4 summarizes how the thesis's aggregated papers contribute to fulfilling each contribution.

Based on our thesis goal, we formulate three main research questions: 1) What are the unique characteristics of ML experiments? 2) What are the overview attributes of ML ExMTs? 3) How can we unify ExMTs and traditional SE tools?

In the following sections, we list and describe our research questions together with sub-questions focusing on specific aspects of interest.

1.2.1 RQ1: What are the unique characteristics of ML experiments

To enable an understanding of the ML experiment and its associated challenges, as well as the common ML project types, their development activities, and evolution patterns, we defined the following sub-questions.

Challenges of ML Experiment Management

RQ1.1: *What are the challenges associated with ML experiment asset management?*

With this question, we seek to understand practitioners' challenges when performing ML experiments without using specialized management tools. Since there are several aspects and activities of ML development, the scope of this question is specifically related to the management of assets such as models, datasets, and metadata during model experimentation or prototyping. For instance, we are interested in the challenges and pain points of using manual or ad hoc approaches to manage experiment assets. Previous work has established and presented technical challenges such as data management, integration, and organizational issues resulting from a disconnect between software engineers and data scientists [4, 8, 9, 22, 80, 81]. In particular, our interest is in the challenges experienced due to ML experiments' unique development characteristics and nature (i.e., the exploratory nature of ML experiments) and how they complicate development tasks. For example, when running *long-term experiments*, what are the challenges of handling assets as new versions are created with the increasing number of experimental runs?

This question provides empirical evidence of experiment management challenges and highlights the need for better tooling support. It also investigates and provides knowledge on specific challenges that researchers and tool developers should address.

Insights into ML-Related Projects: Their Types, Development Stages, and Evolving Patterns

RQ1.2: *What are the ML project types, their development stages, and evolving patterns?*

In order to develop more effective tools and innovative ideas for managing ML development artifacts and processes, it is crucial to have a clear understanding of the different types of projects, the development activities involved, and the patterns that emerge during the transition between workflow stages. With this question, we identify common ML-related project types, and the most frequently performed activities during development, as this can provide insight into the composition and usage frequency of different assets and their respective usage patterns. By answering this research question, we can identify common practices and issues in organizing ML experiments, guide future tool development, and conduct further research. Additionally, this research question aims to track the evolution of development stages through source-code repository commits. This can offer insight into typical developer activities and project evolution patterns, such as whether projects tend to have a short-term or long-term sequence of development activities. Answering this research question will help us better understand and improve ML experiment development processes.

1.2.2 RQ2: What are the Overview Attributes of ML ExMTs

To gain empirical-based knowledge on the features of ExMTs, their benefits and challenges, and their effect on user performance, we proposed the following sub-questions.

Features of ML ExMTs

RQ2.1: *What are the commonalities and variabilities of the features found in existing ML ExMTs?*

As indicated previously, several existing tools aim to support practitioners in asset management when performing ML experimentation. While many tools have a strong target for data science practitioners, it is essential to establish the existing tooling landscape and leverage their support toward new and improved asset management tools with native support for SE and data science practitioners. This research question aims to identify the features found in existing ML ExMTs. For instance, the research question seeks to identify the ML-specific asset types supported by the tools, the asset collection methods, asset storage, and the asset operation adopted by the tools. In addition, this question also explored and compared the features found in the tools sampled from two specializations—i) proposed tools and prototypes from research, and ii) tools used in practice. This question increases our empirical knowledge and understanding of the current solution space for ML asset management, specifically ML experiment asset management. It also provides an understanding of tools' features and the support themes in ML experiment asset management.

Benefits and Limitations of ExMTs

RQ2.2: *What are the benefits and limitations of ExMTs?*

To further increase our knowledge in this subject area, we investigate the gaps in existing tools. This research question seeks to establish why practitioners adopt ExMTs as we elicit the tools' perceived benefits. Similar to the established empirical data on the challenges of managing assets of ML experiments without specialized tools (RQ1.1), we investigate the limitations and challenges of ExMTs as experienced by practitioners. In addition, we explore the barriers limiting the adoption of ExMTs from practitioners who perform ML experiment but fails to employ these specialized tools. With this

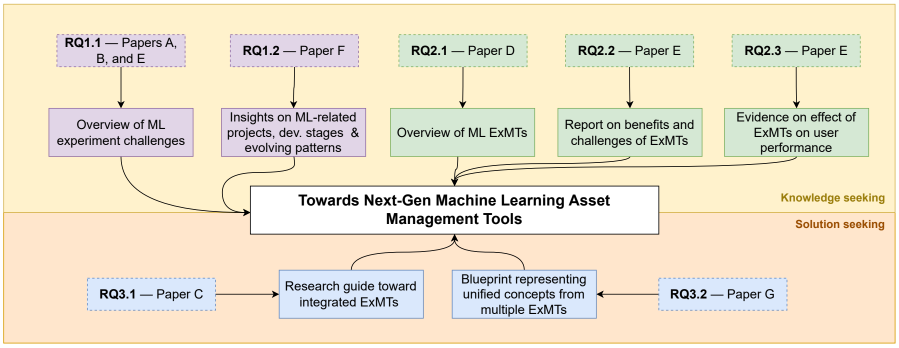
RQ1.1 — Papers A, B, and E
RQ1.2 — Paper F
RQ2.1 — Paper D
RQ2.2 — Paper E
RQ2.3 — Paper E
Overview of ML experiment challenges
Insights on ML-related projects, dev. stages & evolving patterns
Overview of ML ExMTs
Report on benefits and challenges of ExMTs
Evidence on effect of ExMTs on user performance
Towards Next-Gen Machine Learning Asset Management Tools
Knowledge seeking
Solution seeking
RQ3.1 — Paper C
Research guide toward integrated ExMTs
Blueprint representing unified concepts from multiple ExMTs
RQ3.2 — Paper G

Figure 1.4: Summary of paper contributions towards the goal of the thesis.

question, we establish and provide insight into the limitations and challenges of managing experiments with existing tools and potential adoption barriers. Improving our empirical understanding through this question is essential in providing additional requirements for researchers and tool developers.

Effectiveness of Experiment Management Tools

RQ2.3: *How effective are ExMTs on user performance?*

Existing studies have compared the features found in ExMTs [27,32,82,83]; however, none has investigated the effectiveness of ExMTs concerning the impact on user performance when performing ML tasks. To facilitate the improvement of new and improved tools, it is essential to establish the efficacy of existing ones in improving overall development performance. With this question, we establish whether the ExMTs' assistance is valuable enough to motivate their adoption. For instance, this research question probes how asset management via ExMTs differs from the baseline of manual or ad hoc approach. The question investigates the ability to correctly track and fetch assets based on fact-based questions commonly associated with post-ML experiment analysis. The findings based on this question provide empirical proof of why ML ExMTs are essential to improving the development of ML-enabled systems.

1.2.3 RQ3: How can we unify ML experiment management and traditional tools?

To guide the use of relevant empirical findings in ML experiments and ExMTs toward developing new and improved integrated data science and SE tools, we defined the following sub-questions.

Research Steps Toward Integrating ML ExMTs into traditional SE tools

RQ3.1: *What are the necessary research steps toward integrated ML ExMTs?*

Towards a long-term goal of building novel tools that uniformly and natively support software and ML assets management, this research question helps us compile the essential research steps and methodologies for integrating ML ExMTs into traditional SE tools. We conceive and present practical methods for researchers to adopt for tool assessment. For example, the appropriate assessment and the supported operations of management tools that such evaluations should target. Answering this question offers a formal research plan that tools developers and researchers can utilize to improve ML ExMTs towards increasing their value for both ML and software engineers.

Unifying Concepts from Multiple ExMTs

RQ3.2: *How can we unify concepts of multiple ExMTs?*

Our final research question involves effectively representing blueprint and prototype artifacts for new and improved ExMTs. As a further step towards integrating ExMT features with traditional software tools, we seek to explore ways to unify concepts found in multiple existing tools while abstracting their standard features. With this question, we explored how we can represent the structures and concepts supported by existing tools in a common reusable artifact. Answering this question offers the foundation for the unified treatment of ML experiment management and its tools.

1.3 Methodology

We employed five primary research methods to address our research questions as follows.

1.3.1 Exploratory Studies

We performed two types of exploratory studies. i) An exploratory study on performing supervised ML experiments to gain experience in the ML domain and acquire insight into ML experiment challenges and pain points. ii) A large-scale exploratory case study where we adopted mining software repository (MSR) methods [84–88] to analyze ML-related projects. We describe these exploratory studies below.

Exploratory experiments. Our research began with seeking ML experiment hands-on experience on applied ML in the context of pervasive computing [89, 90]. This step provides single-point empirical data on the challenges associated with ML experiment management without specialized tools (RQ1.1: *What are the challenges associated with ML experiment asset management?*). Specifically, our first exploratory study involves the application of supervised ML to forecast building energy consumption in a DHS. While our forecast method presents a new approach for forecasting time series data in building energy [91], the goal of the exploratory study concerning this thesis is to gain practical experience with ML model experiments and identify encountered pain points. We used ML methods through data analysis and modeling for the exploration study. The data analysis and exploration were preceded by domain—district heating systems [92, 93]—knowledge acquisition, followed by data design and collection from ten district heating substations. Employing suitable sensing devices, we continuously and unintrusively collected data from each substation over seven months. The data attributes defining features from the applied domain include the thermal load values, flow rate, supply temperature, return temperature, and weather temperature. With the exploratory data analysis— a critical aspect of data science and model experimentation—, we identified the data features to consider during the ML model experimentation. The objective of the experiment was to accurately forecast the combined space and water heat load used in commercial and residential buildings. Consequently, we performed model experiments over four months involving feature engineering and modeling supervised algorithms to investigate their resulting model prediction accuracy. We experimented with various parameter combinations, algorithms, data partition, and forecast horizons. We performed these modeling experiments in iterations over a long period without supporting asset management tools. Therefore, we encountered several asset management challenges, leading to our single-point empirical data on the challenges associated with ML experiment management without specialized tools.

Exploratory case study. An exploratory case study is a vital empirical methodology to investigate or gain insight into target phenomena in their context [94]. Consequently, we used an exploratory study methodology to perform a large-scale study on the ML-related project types, development stages, and evolution (RQ1.2: *What are the ML project types, their development stages, and evolving patterns?*).

To address this research question, we systematically selected and explored ML projects hosted on GitHub to analyze their types, development stages, and evolution patterns. Our final subjects are GitHub projects dependent on and using one of the top ML Python libraries—*Scikit Learn* and *TensorFlow*. We considered i) projects implemented in Python, ii) projects dependent on either SciKit Learn or TensorFlow library, iii) projects with a minimum of 50 commits, and iv) original projects that are not forked from another project. We excluded projects with cloning errors and those with no identifiable or established stages of the ML development workflow. We obtained 31,066 final projects, which we considered as subjects in our large-scale study.

We performed the study in three different aspects based on our goals. First, we were interested to learn about the types of ML-related projects. For this, we manually analyzed a smaller, random sample of the whole dataset, to identify and define the different types of projects we found and to learn about their prevalence. Second, we aimed to better understand which stages of ML development [33, 95] are maintained in repositories. For example, some repositories may involve data processing or modeling stages only, while others may build on pre-trained models to focus only on the prediction stage. We created and used an API dictionary to map ML-predetermined library APIs to development stages. For example, we parsed the code contents into abstract syntax trees for a specific system and its files. We extract the ML stages available in each project from the obtained tree and map its API calls to our dictionary. Third, we quantitatively analyzed how typical ML-enabled projects evolve, focusing on the different stages identified. We observed the delta between successive commits for each project to investigate how each changes over time. Precisely, we assigned respective ML stages to successive commits by labeling them according to API mappings of relevant calls in the commit deltas.

1.3.2 SLR & Feature-based Survey

Systematic literature reviews (SLRs) are valuable for synthesizing scientific literature on specific topics [96, 97]. Similarly, feature-based surveys are domain analyses carried out to identify the characteristics of specific application domains [98, 99]. Consequently, we employed these two methods to understand the landscape of existing ML ExMTs and the features they offer (RQ2.1: *What are the commonalities and variabilities of the features found in existing ML ExMTs?*).

We aim to understand how ML ExMTs work and their standard features. In addition, we also investigated how tools from the literature compare with those used in practice. To this end, we systematically selected tools proposed in literature and tools used in practice. Our subject selection sources include knowledge (i.e., known research papers from our experience), snowballing, literature database search, and Google search. Using our defined selection criteria, we obtained 12 tools from literature representing state-of-research and 18 tools from grey literature representing state-of-practice tools. Our analysis resulted in feature models representing the variabilities and commonalities across these subject tools. The analyses were primarily based on literature and tool documentation for respective tools. First, we analyzed a single state-of-practice tool to identify its supported ML asset types, collection methods, storage options, and supported asset operations. This step resulted in a baseline version of our feature model. We then iteratively evaluated additional subject tools while modifying terminologies and the model structure to accommodate variations from the new tools being assessed.

1.3.3 Practitioners Survey

Practitioner surveys are valuable for eliciting information about the user experience with software tools or products. We adopted the practitioner survey method to i) elicit information on the challenges of ML experiments (RQ1.1: *What are the challenges associated with ML experiment asset management?*) and ii) gather information on practitioner experience with existing ML ExMTs. For example, what are their perceived benefits and limitations? (RQ2.2: *What are the benefits and limitations of ExMTs?*). With this method, we aim to provide a comprehensive expert survey report on this subject's state of the art and identify areas where improvements are needed. Consequently, for our survey design, we used open-ended, Likert-scale, and multiple-choice survey questionnaires to answer questions on i) the kinds of experiments conducted and what ExMTs and features are used. ii) the perceived benefits of using ExMTs, iii) the limitations and adoption barriers of ExMTs.

We recruited 81 participants in three different batches. First, we recruited practitioners during a regional industrial ML conference and followed up with an email invitation to participate in the survey three weeks after the conference. From this batch, we obtained 24 total participants. Second, participants were recruited via GitHub and identified by filtering for recent projects with dependencies on the top two ML libraries. We ensured relevance of projects by only selecting ones that utilize a defined set of library methods at least once. After that, we randomly fetch contributors to projects with commits lower than 60 to potentially obtain users who are not entirely reliant on Git for asset management. We sent invitation emails to the contributors and got 25 participants, with a response rate of about 1%. Third, participants were recruited via a freelancing service website. To ensure quality, we accepted participation only after reviewing their profiles and asking controlled questions to establish their qualifications. We accepted participation from roughly 60% of the interested freelancers, giving us 32 participants. We employed descriptive statistics and thematic encoding to analyze the obtained quantitative and qualitative responses. For the thematic encoding, we identified recurring and essential themes in the participants' responses and organized these themes in a hierarchy.

1.3.4 Controlled Experiments

Empirical studies, often in the form of controlled experiments, have been widely adopted in SE research to evaluate the values of new SE tools. [100, 101]. In controlled experiments, researchers establish specific conditions (e.g., treatments) under which certain outcomes of interest are observed. There are three important considerations when conducting a controlled experiment: i) the selection of experimental conditions, ii) the assignment of participants to treatments, and iii) the monitoring and measurement of outcomes. In this thesis, we adopt controlled experiments to test the hypothesis of the effectiveness of ML ExMTs on users' ability to improve asset management (RQ2.3: *How effective are ExMTs on user performance?*). Specifically, we wanted to know how adopting ML ExMTs affects user performance. To this end, we recruited fifteen undergraduate student developers majoring in SE to participate in the experiment. Our choice of students as experiment participants was based on several studies [102–107] that suggest that students are adequate stand-ins for practitioners in SE-based studies. We selected participants familiar with ML and using popular ML frameworks and those without experience with ExMTs. Since there is a wide range of ExMTs, it is impractical to consider them all in a single experiment; consequently, we carefully chose two mature and representative example tools with different approaches to tracking, querying, and retrieving assets. We chose a tool representing (i) the intrusive API-based paradigm of tracking assets and the Web dashboard (GUI) paradigm for post-experiment analysis, ii) We chose a different tool representing the CLI-based paradigm of asset tracking and CLI-based post-experiment analysis. As a baseline for comparison, we consider the No-Tool setup, that is, the case of adopting ad hoc strategies without special management assistance from a tool.

We designed a comprehensive experiment to collect participants' experiences using the two described subject tools and ad hoc strategies. Our experiment is based on supervised ML tasks, such as feature selection and engineering, parameter tuning, and evaluation with different learning algorithms. After performing the tasks, participants were asked factual questions based on the generated assets during the experiments. To this end, they used the subject tools to query and retrieve specific data from previous runs (a.k.a. post-experiment analysis). For the No-Tool setup, users were free to adopt any manual or ad hoc strategy but were not allowed to redo or cheat to answer.

We improved the validity of our experiment by adopting a cross-over design, where we divided our participants into three groups with varying orders of treatment. The independent variables in our experiment are the subject tools and the datasets. The dependent variables are the error rate and completion rate of the factual questions posed to the participants. The error rate indicates the number of wrong answers for each subject tool, and the completion rate indicates how many questions were answered for each subject approach. We used the dependent variables to determine the effectiveness of the tools on user performance. The additional dependent variables used under this methodology are the participant's opinions on using the tools and tool paradigms. These variables captured various quantitative and qualitative data based on follow-up questions on the experiment.

1.3.5 Design Science

When seeking new ideas or techniques that drive research development, design science methods are commonly adopted to operationalize research toward new artifacts or recommendations. The process begins with identifying a problem or need and evaluating potential solutions. A design plan is developed to produce an artifact based on the ideal solution. Finally, the artifact is tested and modified as needed. For this thesis, we employed design methodology in three stages: i) Initial design, ii) refinement, and iii) validation. We adopted the design research methodology [108, 109] to address our research question on how to unify concepts found in multiple ExMTs (RQ3.2: *How can we unify the concepts of multiple ExMTs?*).

We aim to establish a reusable artifact representing concepts from various analyzed ML ExMTs. We explored the versioning support offered by 17 ExMTs based on the selection from previous work. We observed and extracted the ML asset types (structures) they support and their versioning relationship. We then unify their conceptual structures and relationships using a metamodel.

Our analysis focused on persistence and versioning support from the tools. We carried out a domain analysis, which resulted in a metamodel, EMMM, representing all structure and versioning relationships found in the subject of the tools. The concepts and relationships were formulated based on a detailed manual analysis of the subject tools and their supported features for asset management. In line with meta-modeling in model-driven engineering [110]—reducing information complexity by abstraction, when we observed conflicts between observations in multiple tools, we chose a higher-level concept that encapsulates them.

The three main stages followed under this methodology are: i) An author performed the initial design of the metamodel to establish its classes and their relationships. ii) We adopted an iterative process to refine the class relationships from the initial design. iii) We performed a validation phase where we populated our metamodel with concrete experiment information from actual experimental revision histories to reveal design flaws and identify improvement opportunities. From the validation phase, we fixed identified design flaws by refining the metamodel from step ii.

1.4 Contributions

This section summarizes the contributions of this research arising from the three primary research questions we described in Section 1.2. In addition, we indicate how our contributions are related to the appended publications of this thesis. Table 1.4 shows the overview of our contributions and their respective research questions and publications.

Our research aims to facilitate the development of next-generation tools for managing development assets for ML-enabled systems. The core of our research centers on ML experiment management—an asset management category for ML-enabled systems. Consequently, we addressed the research aim in two major stages: 1) by establishing and improving the empirical understanding of the state of research and practice concerning ML experiments and ExMTs; and secondly, 2) by proposing research direction and artifacts towards integrating ML ExMTs with traditional SE tools. To this end, the thesis completes the following seven research contributions: i) Presents the challenges of managing ML experiments without specialized tools. ii) Present insights into development stages and evolving development patterns in ML experiment projects. iii) Presents an overview of the state of practice and research on ML experiment asset management tools and the variabilities/commonalities of their features. iv) Provide an empirical-based report on the benefits and limitations of ML ExMTs. v) Present evidence of the effectiveness of ML ExMTs on user performance. vi) Propose a research guide toward integrating ML experiment management into traditional SE tools. vii) Propose a blueprint to unify and represent concepts from multiple ML ExMTs.

Table 1.1: Overview research contribution of this thesis

Contribution	**RQ**	**Method**	**Paper**
Knowledge-Seeking Contributions			
Contribution 1: *Overview on the challenges of managing ML experiments*	RQ1.1	Exploratory study + Practitioner survey	Paper A,B & E
Contribution 2: *Insight on ML-related project types, their development stages and evolving patterns*	RQ1.2	Exploratory Study	Paper F
Contribution 3: *Overview of existing ExMTs*	RQ2.1	SLR + Feature-based survey	Paper D
Contribution 4: *Report on challenges, benefits of ML experiments tools*	RQ2.2	Practitioner survey	Paper E
Contribution 5: *Report on the effectiveness of ML experiment tools*	RQ2.3	Controlled Experiments	Paper E
Solution-Seeking Contributions			
Contribution 6: *Research agenda towards unified and effective software engineering and ExMTs.*	RQ3.1	—	Paper C
Contribution 7: *Blueprint to a unified ML ExMTs*	RQ3.2	Design Science	Paper G

Our first thesis contribution provides the foundation for the rest of our research, where we established the challenges of managing non-traditional assets during and after ML experiments. Our second contribution investigates the characteristics of ML experiments, with details on how they are structured, the predominant activities involved, and how such projects evolve during development. The outcome of our first and second contributions led to our third contribution, where we investigated existing tools and their features aimed at supporting developers during ML experimentations. As our fourth contribution, we elicited empirical data on the benefits and limitations of existing tools from users' perspectives to understand the current gaps that should be addressed in future tools. For our fifth contribution, we investigate the effectiveness of ExMTs regarding how well they support experiment tasks. Our sixth contribution provides a research roadmap toward integrating ExMTs into traditional SE tools. In our last contribution towards integrating ML ExMTs into traditional SE tools, we proposed a meta-model artifact of a unified ML ExMT, which can be used as a blueprint for new tools.

The following part of this section summarizes our contributions extracted from their respective research work and publications.

1.4.1 Contribution 1: Overview of the Challenges of Managing ML Experiments

The development of ML-enabled systems relies on experimentation to optimize algorithms' performance and design. However, managing ML experiments can be challenging, primarily when performed without specialized asset management tools. Challenges are development aspects that make asset (i.e., artifact and metadata) management difficult. To provide empirical data on the challenges of ML experiments, we began with a single empirical data point from our experience (Paper A & B), then broadened the empirical data points using a practitioner survey eliciting data from multiple participants (Paper E).

We addressed this contribution by answering the following research question:

RQ1.1: *What are the common challenges associated with asset management during ML experiments?*

In papers A and B, we conducted an exploratory study with hands-on experience in ML experiments. Here, we experimented with different supervised learning algorithms, including SVM, Linear Regression, FFNN, and Regression Trees, to forecast heat energy consumption at building substations for a DHS network. The study provided a single data point of response to our research question (RQ1.1). The ML experiments were managed without specialized asset management tools and handled by a developer responsible for the experiment tasks, from data preprocessing to model evaluations using ad hoc and manual methods. The datasets for the experiments were obtained non-intrusively from ten building substations—five commercial and five residential buildings—over seven months. The experiment aimed to find the optimal heat-load forecasting models for each building substation.

In the research paper titled E, we conducted a survey with a total of 81 practitioners, out of which 75 perform ML experiments. The survey questions were designed to gather information about the nature of the ML experiments they conduct and the challenges they face while conducting the experiments, with the aim of expanding our understanding of management challenges. We obtained information about the nature of the ML experiment performed from participants who conduct experiments, while the questions on management challenges of ML experiment were directed towards those who do not use specialized tools. Specifically, we requested 23 participants who do not use specialized asset management tools to share their challenges in data preprocessing, model building, and evaluation. In addition, we introduced and described ExMTs to the practitioner and asked them if they thought specialized ExMTs could improve their development process. Using thematic analysis, we analyzed common themes from the responses and compiled them into a report that provides insights into the challenges faced when managing ML experiments. This contribution also provides insight into the nature of the ML experiments practitioners perform.

Results. The challenges of ML experiments are multifaceted and can span many aspects of ML development stages, prominently areas such as data preprocessing, modeling training, and evaluation. The challenges are often non-trivial and compound as experiments grow in size and complexity. We briefly describe the identified challenges established under each of the studied scenarios.

The practical challenges from our *exploratory study* include the following: i) Difficulty in maintaining consistent project organization and structure: Due to the exploratory nature of ML experiments, it took a lot of work to adequately anticipate an ML experiment's evolution. For instance, we could not predetermine how many experiments runs it would take to yield the models with the desired performance. Consequently, the experiment process lacks consistent structure or asset organization for the data and asset types used during our experiments. ii) Difficulty performing domain-specific operations: Although it was sometimes possible to navigate and find specific assets or experiment information based on ongoing development needs, it often requires tedious and several extra steps. For example, typical domain-specific operations during ML experiments are concerned with finding assets based on the state of a specific run (e.g., model performance). This was a significant challenge since manual management approaches do not offer such asset-specific operations. iii) Difficulty effectively addressing experiment concerns: Similar to domain-specific operations, there are common experiment concerns typical for ML experiments. These include reproducibility, traceability, and replicability. We were constantly faced with challenges addressing these concerns because there was no consistent tracking or storage of asset versions as the ML experiment evolved. iv) Difficulty in collaboration and asset sharing: Similar to how version control tools support collaboration in traditional SE, the lack of similar support for ML-specific assets such as learning models introduces a challenge for multiple developers to collaborate on ML projects. v) Difficulty analyzing, interpreting, and comparing the performance of multiple experiment runs. All post-experiment analyses had to be done manually and were time-consuming and error-prone.

From our *practitioner survey*, we elicit information about the nature of ML experiments practitioners perform; 41% of respondents indicate they perform only manual experiments, where the outputs of each experiment run (model

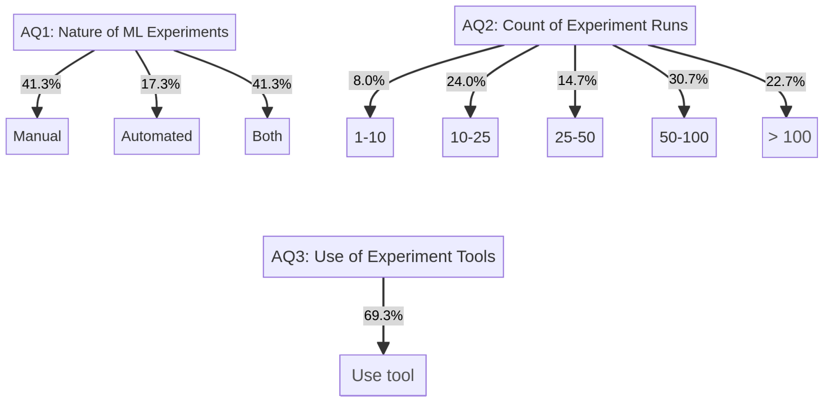

Figure 1.5: Nature of ML experiments.

training) were analyzed and evaluated before deciding on the necessary modifications for the next experiment run. In contrast, 17% indicate they perform only automated experiments using training loops to find optimal results, while 41% of the respondents perform both automated and manual experiments. On the largest count of experiment runs ever performed, 8% of the respondents reported having performed 1–10 runs, 24% reported between 10–25 runs, 15% reported 25–50 runs, 31% reported between 50–100 runs, while 23% reported more than 100 runs. The majority, 69%, of participants in fact use ExMTs to manage assets. Figure 1.5 summarizes the nature of ML experiments conducted by survey participants.

Regarding ML asset management challenges when conducting ML experiments, the common challenges they face when managing ML assets *without* using the specialized tools are: i) inability to ensure essential experiment outputs and their version are consistently and correctly stored, leading to unknowingly overwriting essential assets. ii) difficulty retrieving multiple models and corresponding asset versions from previous runs for reuse, especially in projects with many experiments. iii) difficulty tracking all changes and operations performed on specific assets over an extended period. iv) difficulty interpreting results due to the lack of visualization to correlate dynamic assets to model performance or generate reports to compare different experiment runs. For the question eliciting practitioners' opinion on ExMTs, 80% *strongly agreed* or *agreed* that specialized ExMTs can improve asset management, while only 17% were *neutral*.

In summary:

RQ1.1: Challenges of ML Experiment Management

Challenges make artifacts and metadata management difficult during and after ML experiments. Using an exploratory study based on applied ML in DHS and responses from 23 practitioners, we investigated the challenging aspect of asset management during ML experiments or model prototyping without specialized tools. Our investigation indicates that practitioners face the following challenges:

- *Difficulty carrying out domain-specific operations and maintaining consistent project structure & organization.*
- *Inability to capture all critical state of experiment assets at critical milestones.*
- *Difficulty retrieving multiple models and their linked assets from earlier runs for reuse*
- *Inability to correctly and effectively analyze or interpret results from multiple experiment runs*
- *Difficulty in tracking modifications affecting specific assets over time.*
- *Difficulty in collaboration and sharing of experiment assets.*
- *Inability to effectively address typical experiment concerns such as reproducibility and traceability.*

Based on the nature of the ML experiment conducted, it was observed that over 80% of the participants conducted manual experiments. The majority of the participants conducted between 50 to 100 experiment runs per project. Additionally, almost 70% of the participants used specialized ExMTs to mitigate asset management challenges. Interestingly, 80% of the participants who did not use specialized tools believed that such tools could address the limitations they faced and improve their development process.

Discussion.

In ML experimentation, the persistent challenge revolves around the intricate and time-consuming nature of manually overseeing an expanding array of asset versions stemming from numerous experiments or experiment runs. This challenge often manifests in the form of complexities and substantial time overhead.

Surprisingly, some ML practitioners have yet to embrace the adoption of dedicated asset management tools during the ML experimental or prototyping phases. However, our research findings unequivocally demonstrate that the absence of such support systems introduces a host of noteworthy difficulties. Notably, the prevalence of manual ML experimentation persists, emphasizing the critical need for specialized management tools as a means to enhance both development cost-efficiency and development timelines.

It is worth noting that while conventional tools like VCS find widespread usage as substitutes for specialized ExMTs, there is significant untapped potential for them to effectively address the complexities of ML asset management.

One promising avenue for improvement involves capturing comprehensive asset snapshots, operations, and decision-making processes on a per-experiment-run basis. This approach ensures the holistic traceability and management of ML assets, offering a potential solution to the identified challenges.

1.4.2 Contribution 2: Insights on ML-related Projects, Development Stages, and Evolving Patterns

To build better tools for managing ML assets, we need to improve our empirical understanding of ML-related project development and the common *properties*, *asset types*, *development stages*, and *transitions* between these stages that are involved. Recent studies have investigated the features and support offered by emerging ML asset management tools [27, 32, 77, 78, 82, 83] and attempted to characterize ML projects and experimentation activities (i.e., workflow) empirically [88,111–113]. However, the body of knowledge on the characteristics and practices related to real ML-related projects is sparse. Early studies exist on specific aspects, such as code styles, on collaboration practices, often using small datasets of real projects. Large-scale studies on larger datasets, investigating general characteristics, identifying the exact stages, and also the history of changes, are still missing. For instance, what types of projects are currently developed? What are the predominant development activities? How do these projects and their assets evolve? We addressed this gap with the following research question:

RQ1.2: *What are the ML project types, their development stages, and evolving patterns?*

In paper F, we contribute in this direction and present a large-scale mining study. We contribute a dataset of 31,066 ML-related projects on GitHub and collected insights on their types, development characteristics, and evolution. Our focus was on Python as the most popular language for ML-related projects, as well as on projects relying on the two most popular ML libraries TensorFlow and/or scikit-learn. We formulated the following research questions to cover this contribution: i) What types of ML-related projects are maintained on GitHub? ii) Which development stages can be found in ML-related projects on GitHub? iii) How do ML projects on GitHub evolve and which practices are applied? We selected projects implemented in Python, dependent on established ML libraries—SciKit Learn and TensorFlow, original (i.e., not forked from another project), and has at least 50 revision history. We excluded projects with errors when cloning, not based on SciKit Learn and TensorFlow, and those without identifiable or established stages of the ML workflow when mapped with our library API dictionary. Our selection process resulted in 21,318 Scikit Learn-based and 13,644 TensorFlow-based projects, with 3,896 projects using both libraries. Consequently, we obtained 36,066 unique GitHub projects as subjects for our empirical analysis.

To understand what types of projects are ML-related, we manually analyzed a randomly selected sample of 100 repositories. We defined the categories during this process as we observed recurring project types. Because it is hard to differentiate these categories, we assigned some projects to multiple categories. We manually reviewed the projects, documenting what strategy and parts of the projects we used to identify the project types. To identify concrete ML development stages in our subject projects, we used the API dictionary defined by Biswas et al. [95], which maps popular ML library calls from source code to the development stages. The API dictionary includes functions from core ML libraries—Scikit-learn and TensorFlow—, and other libraries such as pandas, numpy, keras, theano, and caffee to cover the most popular ML libraries studied in prior work [114–116]. We extracted a list of ML stages implemented in each project file. Based on the extracted development stage information, we investigated which stages are present in each project and how many files are associated with them. We also counted how often multiple stages were combined in a single file and which stages frequently occur together. For insights into the evolution of our subject projects, we observed the delta between successive commits for each project to investigate how each system changes over time. Here, we identify the ML stages modified through each commit of our subject systems, and this process is repeated for the other commits of the system to understand its evolution. We grouped our subject systems into five groups based on commit count to observe evolution patterns across different project sizes.

Results.

In our manual project analysis, we observed that the most helpful assets were the project documentation, but also the project structure and types of files present. In total, we identified 7 distinct categories of projects in our random sample of 100 repositories. i) *Experiment*: This ML project type refers to projects that aim to develop a suitable ML model for a specific application by conducting multiple runs of training, testing, and validation using variations of the hyperparameters, data features, and training procedure. The results of each run usually contain a trained model, its performance metrics, and its predictions on the dataset. Incidentally, we noticed that most experiments appear not to store the results from multiple experiment runs, only committing the latest version of the trained model and associated source code. ii) *Education*: This ML project type includes all content that supports education, like practical examples for university courses, homework solutions, or student projects. It is usually explicitly stated in the project description for which course they were developed. iii) *Tutorial*: This project category is designed to facilitate understanding a specific ML topic. This category shares similarities with education, but with the difference that tutorials are not restricted to formal types of education (e.g., university education) but are often less formal and target practitioners. iv) *Research*: This type refers to projects that accompany research papers. The aim is usually the development of new ML models, training methods, optimization techniques, or applications of ML. Experiments are often conducted as part of these projects, leading to most research projects also being classified as experiments. Some research projects could also be called "system prototypes." We distinguish them from systems by the fact that the main goal of research projects is the accompanying scientific publication, not a possible usage by an end user. v) *System*: This type refers to executable applications designed to provide end-user-oriented functionality. We distinguish systems from libraries, which target developers instead of end-users. ML-enabled systems incorporate a variety of different software assets, both ML-related, such as model files, and non-ML-related, such as configuration files or GUI resources. ML-enabled

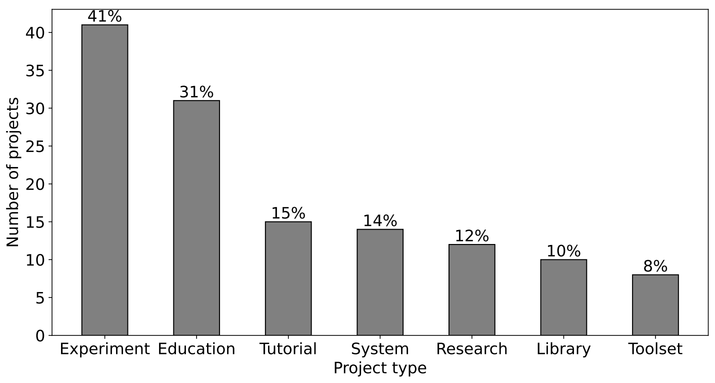

Figure 1.6: Number of projects of each type found in a random sample of 100 repositories.

systems are among the largest and most complex projects in our dataset. vi) *Library*: This category refers to ML libraries, i.e., packaged code to be integrated into other projects—especially ML-enabled systems—via an API. These libraries range from specialized toolkits for ML-related purposes, such as deep learning, computer vision, or natural language processing, to more general-purpose libraries that offer a wide array of algorithms and utilities. The libraries in our sample are often well-maintained with active collaborators. *Toolset*: This type refers to ML-related projects that are intended for practical use but are very limited in their applicability. In contrast to systems, they lack the property of interconnected components and functionalities. Instead, they consist of a set of isolated functionalities. Figure 1.6 shows the prevalence of the different categories found in our sample.

The *ML workflow stage* identified within our subjects include `data acquisition`, `data preparation`, `modeling`, `training`, `evaluation`, and `prediction`. In particular, their prevalence is as follows: we found the data acquisition (30,063) and preparation (31,008) stages in over 96 % and 99 %, respectively. The modeling stage occurred fewer times in 26,808 projects, the training stage in 27,140 projects, the prediction stage in 26,136 projects, and the evaluation stage in 21,894 projects. This shows that most projects involve data acquisition and preparation, while the evaluation stage is the least observed ML stage. We note that the prevalence of these stages is very similar to the results of Biswas et al. [95] obtained on a set of 105 data science projects from Kaggle. For unique combinations of development stages found in each project, we found 17,333 projects (over 55 %) implement all of the six ML development stages. Overall, we identified 51 unique combinations of these ML stages, ranging from projects with only one stage to those including all. Figure 1.7 shows an overview of the most common combinations (i.e., those with at least 100 occurrences). As we can see, there is a larger gap between projects implementing all ML stages (17,333) and every other combination, with the next largest one missing the evaluation (4,196 projects, 13.51 %). Interestingly, only the data preparation stage is present in all combinations, while the evaluation stage is often missing.

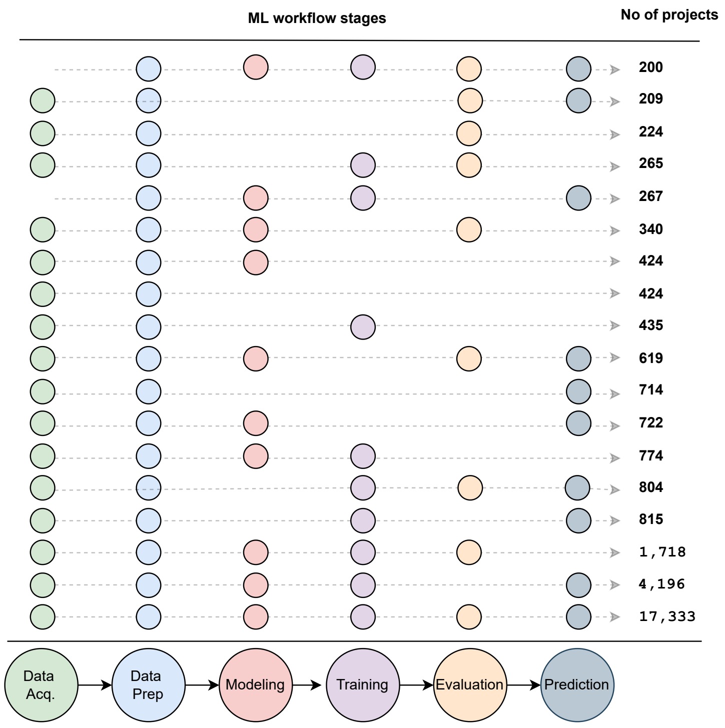

Figure 1.7: Combination of ML workflow stages (freq > 100)

Figure 1.8 presents the proportion of our subjects affecting different ML development stages throughout their lifecycle as observed in their commits, showing their evolution pattern. Our findings show that most project changes predominantly impact the data acquisition and preparation stages. Specifically, approximately 50% of the projects consistently make changes to the Data Preparation stage over time. We observed that aside from the initial few commits, modifications related to data preparation remain consistent throughout the lifecycle of the projects. As for the Data Acquisition stage, on average, over 30% of the projects introduce changes to this stage over time. However, this activity has a slight downward trend as the number of commits increases.

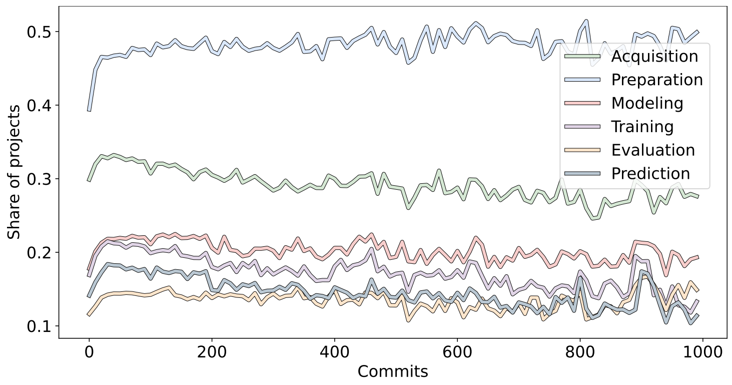

Figure 1.8: Share of all subject projects changing a specific ML stage at a certain point in development.

RQ1.2: Insights on ML Projects, Dev. Stages, and Evolving Patterns

Using a large-scale exploratory study, where we explored 36,019 ML-related projects for insights into their types, development stages and evolution patterns, and we observe the following:

- *ML projects on GitHub are diverse. Majority are non-production-focused, including experiments/research or tutorials/education. Libraries, toolsets, and ML-enabled systems are minorities. Boundaries between types are not always clear, particularly between education/tutorials and toolsets/systems. We relied on a subjective evaluation to categorize the latter.*
- *Most ML-related projects cover all typical development stages. Variations in the covered stages may imply different practices and goals, but the low ratio for evaluations indicates that many projects are exploratory and not focused on training models to be used in practice.*
- *Finally, Data acquisition and preparation remain continuously relevant ML development stages throughout such projects' evolution, requiring corresponding tool support for developers.*

Discussion. We found that ML projects on GitHub span a wide array of different types, with the majority being non-production focused. This indicates a need for further research into how the development of open-source ML projects can transition from prototypes and experiments to mature software systems.

The small fraction of these projects being actual ML-enabled systems raises the following considerations. A reason could be diverse tool dependencies (i): One possible explanation for the limited presence is the complex nature of such systems, which mostly requires developers to rely on an array of specialized tools serving a unique purpose. It is plausible that Git repositories are used to manage a subset of all assets, such as documentation and code, while other assets, such as data, trained models, and so on, are managed through alternative tools. Another reason could be the isolation of ML experiments and software systems (ii): Our findings could also be interpreted to support the theory that ML experiments and software system development are often conducted in isolation rather than as an integrated project. This separation may arise from the historical division between data science and SE teams, resulting in disjoint workflows and project structures. For example, there could be separate teams or developers working on different aspects of ML-based systems. For instance, the ML component could be managed as a Git project by a team while a separate team develops the software systems utilizing the model.

These findings illustrate a need for novel studies that specifically identify ML-enabled systems on a large scale. This, however, requires designing automated project-type identification techniques that can be used to drive future studies. In addition, identifying the reasons behind the limited presence of systems requires exploratory studies, relying on interviews or surveys with developers, also confirming or refuting our results.

Our results also show that the most important aspects of the development of ML-enabled software are related to data acquisition and preparation. This finding confirms that data management is typically identified as the biggest challenge when building software with ML models, such as ML-enabled systems [117–119]. These findings demand improved tool support for DataOps, specifically for making changes to the data-related parts of ML pipelines.

1.4.3 Contribution 3: Overview of existing ExMTs: the State of Practice and Research

Many ExMTs, described in Section 1.1.4, aim to address the challenges of managing ML-specific assets reported in Section 1.4.1. These tools target practical experiment concerns, including reproducibility [26–28] and traceability [31], by providing functionalities to store, track, and version assets from different experiment runs. While these tools have become available recently, they have yet to fully mature, especially compared to their traditional SE counterparts. It is, therefore, essential to assess the support found in the current tool landscape to facilitate further research toward improving them, developing new ones, and consequently improving the engineering processes of ML-enabled software systems. While related works have compared the features of the tools, we present a formal representation of their commonalities and variabilities with the following research question:

RQ2.1: What are the commonalities and variabilities of ML ExMTs?

In Paper D, We survey asset management support found in *state-of-research* and *state-of-practice* in existing and proposed management tools for ML experiments, identifying the types of assets supported and the operations offered to users for managing ML assets. Specifically, we conducted a feature-based survey to identify the characteristics of ExMTs, then modeled the characteristics as features in a feature model [98, 99]. For state-of-research, we conducted a Systematic Literature Review (SLR, [97]) to select the relevant literature and qualitatively analyze it to answer our research questions. For state-of-practice, most tools do not have related scientific publications; consequently, we collected the relevant tools from the grey literature. For both cases, we specified relevant inclusion and exclusion criteria to filter and define the scope of tools we analyzed.

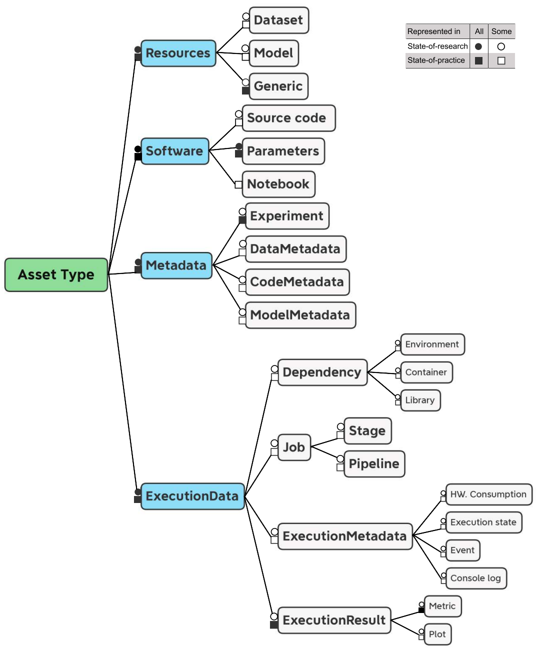

Figure 1.9: Asset Types: A representation of the data types tracked by the subjects under study.

Overall, our contribution comprises:

- Feature models representing the variabilities and commonalities of asset types, collection methods, storage methods, and the operations found in subject tools.
- A comparison between tools features found in state-of-research and state-of-practice.

Result As our contribution, we proposed feature models to characterize and describe the ML asset types and the management support found in our subjects. The top-level features—`Asset Type`, `Collection`, `Storage`, and `Operation`—are discovered as the core features of the subjects in our study. `Asset Type` outlines the data types that are tracked by our subjects; `Collection` describes how the assets are collected; `Storage` explains how the assets are stored and versioned; `Operation` specifies what operation types are supported.

Asset Type: Compared to traditional software development, ML development involves diversified asset types, and the supported types vary across our subject tools, with different levels of support. We define `Asset Type` s as the set of data types recognized or tracked by our subjects. Particular asset types can be supported either explicitly or implicitly. Our analysis identifies features `Resources`, `Software`, `Metadata`, and `ExecutionData` as the sub-features of `Asset Type`. `Resources` represent the core asset types of the ML workflow. In contrast, `Software` refers to the implementation responsible for changing the

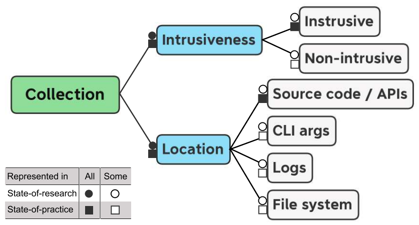

Figure 1.10: Collection feature model: A representation of collection features used to track asset types.

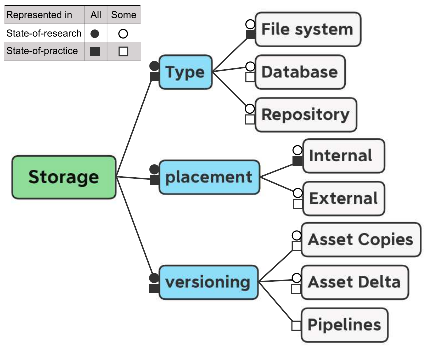

Figure 1.11: Storage feature model: A representation of the storage feature identified in the subjects under study.

states of these assets. The `metadata` are structured descriptive information about `Resource` and `Software`, providing context and details for effective asset management. The feature `Executiondata` represents information about the execution of ML runs and the outcome of such runs. Figure 1.9 shows the feature model representing `Asset Type`

Collection: The feature `Collection`, shown in Fig. 1.10, represents the options provided by our subjects to track their supported asset types. The feature `Intrusiveness` shows the level of the explicit declaration required to collect the assets. The feature `Location` represents the point where the subject tools collect the assets. `Intrusiveness` describes the amount of instrumentation required by users to track assets.

The feature `Storage` describes how the assets are stored and the versioning type supported by the subject tools. Figure 1.11 shows the sub-features of `Storage`. Under `Storage Type`, we identify `File System`, `Database`, and `Repository` as the storage types of our subjects. The `File System` type is the simplest storage type provided by our subjects: tools store collected or tracked assets as objects on file systems. `Storage placement` indicating the location of stored assets relative to the tool can either be `Internal` or `External`. `Versioning` indicates how asset versioning is supported by the tools, which can be supported by: i) storing copies of assets each time they are modified (`Asset copies`), ii) storing the deltas between assets for space efficiency (`Asset delta`), or iii) versioning pipeline metadata to reproduce assets (`Pipelines`).

The feature `Operation` represents the operations supported by our subject tools. The `Track` and `Explore` operations are common features supported by all subjects. The `Track` operation is the core feature offered by the subject tools. Our subject tools track either assets or metadata about them. The `version` operation feature indicates support for versioning-related operations similar to the conventional VCS like Git. The feature `explore` represents operations that help derive insight or analyze assets collected from completed ML experiments. Our subject tools support users in various ways to `Query` assets, from simply listing all experiment assets to advanced selection based on model performance to compare different experiment iterations or runs. `Visualize` indicates the use of graphical presentations (e.g., charts and graphs) of experiments and their associated assets, such as performance metrics at different points in time. The feature `Retrieve` represents the means to retrieve assets from the tools. Feature `Execute` indicates operation support that allows subject tools to manage the execution of ML experiments. Sub-features under this include `Run`, `Reproduce`, `Multistage` and `Distributed`. We represent the following management operations with feature `Manage`: `Modify`, `Archive`, `parameter search`, and `model registry`. Lastly, we identify feature `Collaborate` indicating the presence of collaboration features such as `Share`, `Publish`, `Export`, `Import`, and `Discover`.

For the comparison between state-of-research and state-of-practice, we find that state-of-practice subjects support more asset types than the observed state of research subjects. All the state-of-practice subjects support the *tracking* of generic resources, which implies that they can track arbitrary files. Both state-of-practice and state-of-research subjects provide the option to track parameters or hyperparameters used during ML experiments. While 41% of the state-of-practice tools recognize and support tracking computation notebooks as an asset type, none of our state-of-research subjects provides dedicated support for tracking computational notebooks. Both groups of our subjects rely heavily on metadata describing ML experiments and their associated assets. Although all subject tools support static metadata assets, we observe more metadata types for the state-of-practice subjects. Roughly half of the subjects in each group support the pipeline management features (Section 1.1.4) through the representation of workflows as stages and pipelines. Execution results are supported and tracked by 58% of state-of-practice tools and 50% of state-of-research tools, while fewer subjects track execution metadata. The collection points for both groups are primarily through source code using programming APIs provided by the subjects. In addition, the state-of-practice subjects notably provide alternative asset collection from the command line or instrumented configuration files. A few subjects from both categories also support versioning operations similar to conventional VCS operations. Subjects in both categories offer support to query and visualize assets, with most subjects providing access via web-based dashboards.

In summary:

RQ2.1: Empirical data on ML ExMTs' features

Using SLR and the feature survey method, where we analyzed data from 18 tools used in practice and 12 tools described or proposed in the literature, our investigation indicates the following:

- *The commonly supported asset types are generic files, parameters, experiment metadata, and execution results.*
- *Assets are collected intrusively, primarily through source code APIs, CLI arguments, and configuration files.*
- *The assets are either stored internally or externally (e.g., cloud storage) in file systems, databases, or repositories.*
- *All tools offer asset tracking operation, while 93% allow exploration of assets via queries and visualization.*
- *The tools used in practice and those proposed in the literature support similar asset types, collection methods, storage methods, and operations. A notable difference between them is that the state-of-practice tools predominantly support features associated with other management tool categories, such as pipeline management, execution-related operations (e.g., multistage and distributed execution operations), and collaboration.*

Discussion We found domain-specific operations tailored to ML asset types in those tools that explicitly manage specific asset types. For example, ModelHub [62, 120] offers a domain-specific language to assist users in performing experiment operations (e.g., evaluating a model with a given dataset as input). Domain-specific operations are not widely supported across the subject tools. Similarly, reproducibility is the most addressed experiment concern across all the subject tools. While most subject tools support tracking assets required to reproduce current or previous experiment runs, only about half offer explicit reproducibility operations. Having a domain-specific language that is specifically designed for ML is crucial in providing users with helpful support.

We observe that most subjects' asset collection methods are `intrusive`, i.e., they require users to instrument or modify their source code to track asset information. This method is tedious and error-prone and can also deter the adoption of ExMTs due to the associated overhead cost. Some tools such as ModelKB, MLFlow, and Weights & Biases support automatic asset collection in `non-intrusive` ways to address these drawbacks. However, many tools still only support automatic asset collection for a limited number of popular ML frameworks, such as TensorFlow and SciKit Learn. We believe that further effort on automatic asset collection will assist users significantly by eliminating error risk and reducing development time.

1.4.4 Contribution 4: Report on Challenges and Benefits of ExMTs

The increasing popularity of ML ExMTs has been reflected in the growing number of studies on the subject matter [32, 77]. However, our interactions with practitioners, esp. at an industrial ML conference, reveal that many practitioners who are aware of them are reluctant to adopt them for various reasons. For example, some found them unfit for their way of working—a typical problem for tools. To the best of our knowledge, there are no user-based empirical studies on ML ExMTs, specifically on their actual benefits and challenges. Improving our empirical understanding from this perspective is essential to improve these tools, providing requirements for researchers, tool vendors, and educators.

We addressed this gap with the following research question:

RQ2.2: *What are the benefits and challenges of ExMTs?*

In paper E, we surveyed 81 ML practitioners who i) attended an industry-focused ML conference, ii) made recent contributions to ML-based projects on GitHub, and iii) are relevant practitioners from an online freelancing service. The survey elicits information about the ExMTs used, their perceived benefits, and their challenges and limitations. We also elicit information from practitioners who do not use the ExMTs to understand common adoption barriers.

The following statistics describe the participants of the survey. 35.6% described their role as data scientists, 31.7% as ML engineers, and 12.5% as software engineers, while other indicated roles include data engineers and researchers. The average experience is 4.4 years. 32.2% of the participants indicated technology as their current domain, 17.8% education, 13.6% health, and 11.9% consumer retail. Other domains include consumer retail, telecoms, transport, gaming, and agriculture.

Results.

Perceived Tool Benefits: Most participants perceived ML ExMTs as highly beneficial. 72% of the responses strongly agreed or agreed that tools facilitate their ML tasks, while 18% were neutral. 39% were neutral on the ease of learning and using the tools, while 45% either agreed or strongly agreed to ease of use. 76% strongly agreed or agreed that ExMTs make them perform experiments efficiently, while 12% were neutral. 48% agreed or strongly agreed that using ExMTs helped improve their model performance, while 30% were neutral. 74% agreed or strongly agreed they obtain management benefits when using the tools compared to when not using them, while 20% were neutral. 33% disagreed that simple command-line interfaces similar to Git are sufficient for querying and analyzing tracked experiment assets, while 29% were neutral, with 23% agreed or strongly agreed. 69% agreed or strongly agreed that GUI dashboards are essential for efficient querying and analyses of experiment assets and metadata, while 20% were neutral. 63% prefer or strongly prefer dedicated tools over multi-purpose tools with extended features, 22% do not prefer such, while 16% are neutral. Responses were almost uniform for the benefits and values concerning specific experiment challenges addressed by management tools. In order of popularity, the benefits

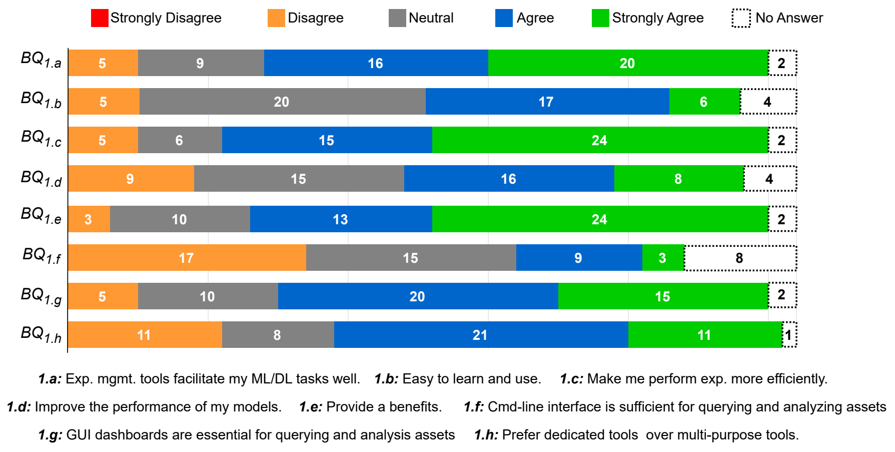

Figure 1.12: Result: questions related to perceived benefits (BQ_1)

are time savings, experiment result analyses and comparison, traceability, reproducibility, result and model optimization, collaboration, and applicability. Figure 1.12 summarizes the outcome of questions related to the perceived benefits of ExMTs.

Limitation and Challenges: When asked if participants experience limitations with ExMTs, 6.7% of our respondents strongly agreed, and 29.3% agreed to experience limitations with the tools affecting their experiments. 50% of the responses were neutral, while 14% either disagreed or strongly disagreed.

The specific issues reported about the tools are technical restrictions, vendor lock-in, computing resource limitations, missing features, usage costs, and a steep learning curve. Our participants reported various technical issues. For example, 15% of the code count from the thematic analysis indicates tool support for a few asset types, while 8% indicates a preference for more flexible and non-restrictive tools with extended support for custom asset types. By design, some tools track assets as immutable objects to ensure persistence; however, some participants indicate this as a limitation. Data accessibility problems were also reported, as some tools do not interface with custom data stores. Our participants also indicated that tools primarily target data scientists and ML engineers and do not fit perfectly into SE workflows. Some tools' visualization features are also reported to be too simple and limiting.

On missing features, participants experienced limitations due to a lack of: automatic parameter search, direct integration with databases, custom ML pipelines support, authorization and authentication support, VCS (especially Git) integration, and integration with post-deployment operations and existing visualization tools. On the cost and computing resources issue, since many tools offer cloud-based SaaS. The services often offer free services with limited computing resources, leaving freemium users with storage, memory, and computing resource limitations. Consequently, practitioners find the service usage cost as a limitation. Another related limitation is the vendor lock-in issue, which makes it difficult for practitioners to adopt tools or services different from their current vendors. For standalone-tool users, some see the restriction to a local machine as a limitation since they cannot take advantage of faster computing resources.

Furthermore, regarding challenges experienced when using the tools, 34% of the thematic code count indicates poor documentation or a steep learning curve as a challenge. 14% indicate the tools lack robustness and consistent availability, making them immature and buggy. For example, a participant reported experiencing strange tool behavior after reaching hundreds of iterations. 14% indicate challenges in tool setup or usage in development team settings where strong collaboration is required.

In summary:

RQ2.2: Benefits and Limitations of ExMTs.

Using a survey with 81 practitioners as participants, where we elicited information on the perceived benefits, limitations, and challenges of ML ExMTs, we obtained the following:

- *Most of our survey respondents find experiment management highly beneficial as they indicate that: i) they facilitate their ML experiment tasks, ii) are easy to use, iii) help them perform experiments efficiently, and iv) help them obtain performing models faster. Overall, 74% of obtained responses indicate the tools offer management benefits when using them compared to when not.*
- *As reported by our participants, the limitations of ML ExMTs include technical restrictions, vendor lock-in, computing resource limitations, missing features, usage costs, and a steep learning curve. The challenges associated with using the tools include poor documentation, lack of robustness, inconsistent availability, making them immature and buggy, and usage complications in team settings where strong collaboration is required.*

Discussion The benefits of using ExMTs are evident among tool and non-tool users. Thus, addressing the highlighted challenges and limitations associated with adopting and using the tools can lead to their uptake among practitioners. Additionally, successful and effective implementation of such tools can help users fully utilize the potential benefits of such tools. We propose the following recommendations for actionable solutions. To address steep learning curves, we propose comprehensive documentation and user-friendly tutorials to facilitate user onboarding. Additionally, intuitive interfaces can empower users to navigate and utilize the ExMTs more effectively, boosting their overall experience. Tool maturity should be a focus for tool developers, enhancing reliability and stability. Ensuring robustness should involve rigorous testing, bug fixing, and robust error-handling mechanisms. We also recommend that ExMTs offer customizable workflows and compatibility with diverse ML frameworks and libraries. Tailoring tools to individual user needs can foster adaptability and seamless integration within existing toolchains. For cloud-based tools, emphasizing open standards and compatibility can reduce the risk of vendor lock-in and empower users with flexibility and control.

1.4.5 Contribution 5: The Effectiveness of ML ExMTs

While related studies have compared the features of ML ExMTs [27,32,77,82,83], none have evaluated the effectiveness of the tools on how they impact user performance. To further improve our understanding of our research's subject, we present the first empirical study on the effects of using ML ExMTs. We investigate these, along with users' preferences on the essential tools' paradigm.

We contribute this additional empirical insight on ML ExMTs with the following research question:

RQ2.3: *How effective are ExMTs on user performance?*

In paper E, we conducted a comprehensive controlled experiment where we guided fifteen student developers with a background in SE to perform typical supervised ML tasks. To improve the validity, we adopted a cross-over design [121], where we divided our participants into three different study groups with 5 participants per group. This design enhances statistical power by abolishing individual subject differences and generating more data points [122]—in our case, it increases the number of data points by a factor of 3. We applied two selection criteria for our recruitment of participants: i) familiarity with ML and popular ML frameworks, such as SciKit Learn, and ii) Participants must not have prior experience with ML ExMTs. We decided to compare the tools' paradigms rather than the tools themselves because i) the outcome of the study can be applied to other tools, and ii) tools for experiment management are currently evolving; thus, the subject tools and their principles and paradigms may evolve quickly. Consequently, we measured the effectiveness of two tools that offer different usage paradigms and compared them to a baseline of not using any such tool and to each other. First, we chose Neptune.ai, representing i) the intrusive API-based paradigm of tracking assets and ii) the Web dashboard (GUI) paradigm for post-experiment analysis. Second, we chose DVC, representing i) the CLI-based paradigm of asset tracking and ii) CLI-based post-experiment analysis.

We guided the controlled experiment participants through some ML tasks with different tool setups. They were later asked factual questions and scored on their ability to answer or complete the set of factual questions correctly. The error and completion rate for questions reflects the effect of the support offered by the subject tools when performing ML experiments. The tools provide users with the option to organize their experiment assets. For instance, there are greater chances of stating wrong answers to factual questions about completed experiments when there is no structure for managing assets. To discuss the value of the subject tools, we calculated the error and completion rate for each tool across all study groups (Fig. 1.13 shows the mean values). The completion rate describes the ratio of attempted questions, while the error rate implies the fraction of wrongly answered to all attempted questions. Lastly, we asked the participants about their opinions and experiences with the tools paradigms.

Results.

Completion and error rates: The responses for when participants were using Neptune have an average completion rate of 98% and an average error rate of 7%. DVC obtains an average completion rate of 96% and an average error rate of 29%. The No-Tool alternative has an average completion rate of 84% and

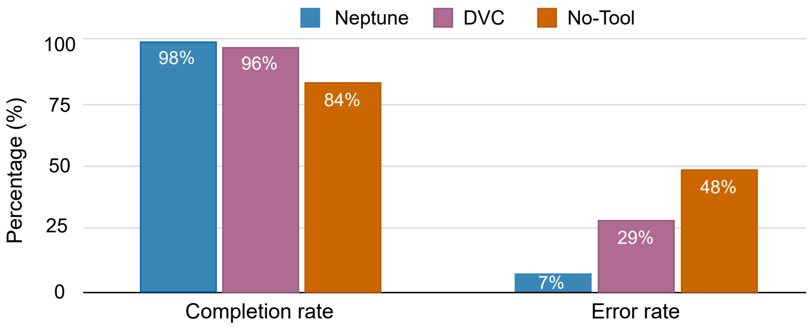

Figure 1.13: Results: Average completion and error rates

an error rate of 48%. The error rate was lowest when using Neptune, followed by DVC, and participants made the most errors when using the 'No-Tool' approach.

To evaluate whether the differences are significant enough, we conducted a Kruskal-Wallis test together with post hoc comparisons with a Bonferroni-corrected significant threshold. We conclude from the analysis that Neptune differs significantly from No-Tool, but we do not find statistical significance in the other cases. For the error rates, there is a highly significant difference between Neptune vs. NoTool, and Neptune also shows a significant difference vs. DVC. However, the error rates of DVC and No-Tools are not significantly different. Interestingly, the perception of the participants was still different: the overwhelming majority found it difficult to complete the tasks without the use of any ExMT.

User's perception of tools: On the ease of completing the tasks, the overwhelming majority found completing the tasks very easy with Neptune. 93% of the participants found it to be *Easy* or *Very easy*. For the same question about DVC, the responses were mostly neutral, with 46% responding *Neutral*, 33% responding *Easy*, and 20% responding *Difficult*. Notably, 80% of the participants found using "No-Tool" *Difficult*. On querying and retrieving assets to compare experimental runs, most participants (93%) found the web-based dashboard of Neptune very helpful, and 53% of the participants were neutral about DVC's CLI paradigm, with 47% finding the CLI helpful. For "No-Tool," 53% thought the manual approach was not helpful, with 20% neutral responses.

On the importance of using ExMTs versus "No-Tool," all participants agreed (80% *Strongly agree*, 20% *Agree*) that the subject tools provide significant support for tracking and retrieving assets during model development. Regarding users' preference for tracking assets among Neptune and DVC, the response is balanced, with 53.3% preferring Neptune and 47% DVC. For the best tool for querying and retrieving previously tracked data, Neptune took the lead with 73% in its favor, while 27% prefer DVC. Also, 73% of the participants prefer Neptune's GUI dashboard for comparison over DVC's CLI commands. For ease of learning, most participants (73%) believe DVC was the easiest to learn. We believe this reflects the experience with CLI-based tools, such as Git, for managing assets in traditional SE. Lastly, 37% reported that Neptune provides the best support for comparing experiment runs.

RQ2.3: Tools' effectiveness and User perception

Using a controlled experiment with fifteen developer students, where they performed ML experiments using three different setups, including specialized tools and manual approaches, we obtained the following:

- *The controlled experiment established that ML ExMTs improve user performance compared to ad hoc approaches to managing ML assets.*
- *On the asset tracking modes, we did not observe an unequivocal preference for either API-based instrumenting source code versus a non-intrusive CLI-based approach. At the same time, participants prefer a GUI-based tool over a CLI-based one for querying and retrieving assets from the management tools.*

Discussion. *Tool Paradigm Comparison:* The preferences regarding asset tracking modalities (API- vs. CLI-based) were balanced. Observation contrasted our expectations because of the drawbacks often associated with the intrusiveness of API-based tracking [55], in particular, the required manual overhead and error-proneness [55, 78]. A possible explanation could be experience or familiarity with CLI-based tools; users who are less comfortable with CLI commands may prefer an API, irrespective of its associated overhead. Another explanation could be that the experiment participants did not consider the extra lines of codes required for instrumenting and tracking assets as an overhead but as part of the necessary tasks, given the experiment's focus on management tools.

Informal Asset Management. Hill et al. [19] found that practitioners managing ML assets rely on informal methods, such as notes, spreadsheets, and emails. In line with this, as an informal way of tracking assets, most participants printed out asset values during the tasks. These informal means of managing ML assets are the only option usually available to users when there is no tool to offer a systematic way. These informal ways of tracking experiment assets are considered expensive, time-consuming, and error-prone [19, 63]. While the informal method of printing out the values might have helped some of the participants answer our factual questions, we expect that the completion or error rates for these questions would be much lower if the users were asked after a week of completing the experiment. Common practical scenarios may take several days or weeks after performing an experiment before such questions arise. When they do, it is crucial to provide accurate answers, underlining the need for a structured and tool-supported asset management approach.

1.4.6 Contribution 6: Research agenda towards unified and effective SE and ExMTs.

ML ExMTs support developers and data scientists to track and retrieve ML experiments and assets when building intelligent software systems. Unfortunately, despite the availability of a large number of these tools, when engineering intelligent, ML-based systems, these tools are not integrated with traditional SE tooling. Hence, there need to be more novel tools providing native support for managing software and ML assets uniformly. We recognize integrated tools as a valid path to unifying the management of assets of ML experiments and traditional software development. We provide the essential steps towards such tools with the following research question.

RQ3.1: What are the necessary research steps toward integrated ML ExMTs?

In paper C, we present a short research agenda toward unified and effective SE and experiment management tools. We advocate for empirical assessments of ExMTs and propose a research agenda that can inspire follow-up work by researchers and tool builders on assessing and improving ExMTs to increase their value for ML and software engineers. A core research challenge towards such improvement is integrating experiment management supports into the traditional SE tooling. To this end, identifying commonalities and differences between the two tool landscapes among different dimensions, including process, organization, technology, and architecture, is needed. The commonalities can be unified into common tools, while differences remain separate or become add-ons. Results from tool effectiveness studies can further steer such integration, especially in identifying essential features and paradigms to support.

Result

The four proposed steps (See Fig. 1.14) are summarized in the following paragraphs:

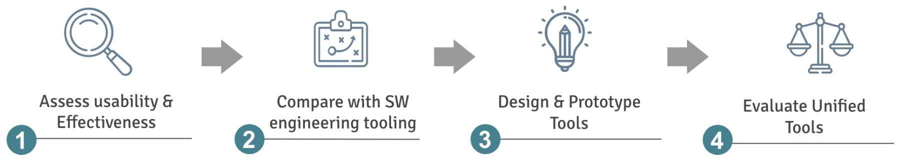
Assess usability & Effectiveness
1
Compare with SW engineering tooling
2
Design & Prototype Tools
3
Evaluate Unified Tools
4

Figure 1.14: Proposed research steps towards integrated ML ExMTs.

1) *Assess Usability and Effectiveness.* To elicit practical empirical data on ML ExMTs, we propose controlled experiments, addressing questions such as: *How do these tools affect user performance? How do users perceive them? What are the effects of the different realizations (paradigms) of the tool features on users?* Tool candidates can be sampled from available tools based on tools' paradigms. For example, when investigating the effect of asset storage paradigms on users, studies must select and classify subjects based on existing storage options in tools. The experiments should consider a special control group that performs the experiment without specialized ExMTs. Experiments focusing on usability and learnability are also important, incorporating participants with different backgrounds and expertise (e.g., experienced vs. non-experienced, practitioner vs. researcher, software engineer vs. data scientist). Experiments can target specific ML experiment concerns. We identify *tracking*, *querying*, and retrieving as basic concerns operations to all respective tools [78]. Consequently, experiments should exercise these base operations in the tasks. Participants should be guided through ML tasks that mimic real ML experiments, which involve multiple experiments and experiment runs evolving incrementally, where participants modify ML assets resulting in new versions. Later in the

experiment, participants should be required to use the tools' operations to (or manually) track, explore, and retrieve experiment runs and assets.

Independent variables that can be used to assess tools under investigation are (i) user performance and efficiency and (ii) user perception. For the former, we propose to measure with the metrics *completion rate* and *error rate* of tasks, as well as *response time* to answer factual questions. These can be elicited with a questionnaire incorporated into the experiment guide. For the independent variable based on user perception, essential for tool adoption and elicits subjective user opinions on tools and their features, metrics should measure participant ratings on the ease of completing the tasks with each tool or the level of support offered for tracking, querying, and retrieving ML assets. Results can indicate the essential tool features. In addition, qualitative, open-ended questions complete the picture. Responses might report specific tools or features that are difficult to use, indicating poor usability.

2) *Compare with SE Tooling.* Studies should determine commonalities and differences between ML experiment management and traditional development tooling, addressing, for instance: their *common features* or *common workflows.* We advocate feature-based surveys, relying on a domain analysis technique, often conducted for similar comparisons, including special kinds of VCS [123]. Studies need to focus on the problem space (required activities and workflows of users) and the design space, architecture, and offered operations.

3) *Design and Prototype Unified Tools.* As a next step, relying on the results of the prior ones, we propose creating meta-models representing the tools' conceptual structures (for example, assets, relationships, and versioning), unifying the studied tools. Ideally, the meta-models are supersets, customizable towards specialized tooling based on prospective users and usage context in the future. For instance, add-ons can provide dataset-specific views or features not required in other tool instances.

4) *Evaluate Unified Tools.* The resulting prototype should be evaluated with user studies, including experiments, simulations, action research, or surveys. Effectiveness and usability are important claims to evaluate, but also scalability and learnability. Unified tools should not compromise compared to the standalone ExMTs. The evaluations can reuse the methods from step 1 (*Assess Usability and Effectiveness*) but should combine data science and SE activities to evaluate the effectiveness of the unification. Users' qualitative perceptions of the new tools can also be elicited through surveys, but we believe action research to be especially fruitful in understanding user interactions and tools' benefits.

1.4.7 Contribution 7: Blueprint to a unified ExMTs

Our previous contribution identifies a path toward integrated ML management tools that require them to be built on traditional ones and extended with domain-specific operations tailored to ML assets. Such tools should ideally be interoperable with existing ExMTs. Furthermore, developing such tools should incorporate the domain knowledge about ML experiment management found in existing management tools. Hence, it is essential to investigate how we can represent a unified instance of existing tools. We contribute to this path by answering the following research question.

RQ3.2: *How can we unify the concepts of multiple ExMTs?*

In paper G, we propose the *Experiment Management Meta-Model (EMMM)*, a metamodel that unifies concepts and relationships extracted from systematically selected ExMTs, focused on the concept of *experimental runs*.

Our metamodel characterizes two main concerns: (i) ML asset structures as concepts and their relationships as observed in the state-of-the-art tools; (ii) conceptual version control structures that can hold ML and traditional assets. For these concerns, we performed a domain analysis, in which we developed *EMMM*, representing the superset of the concepts supported by considered subject tools. The two main components of the metamodel are classes and references showing the concrete concepts and relationships in ML ExMTs.

We formulated these concepts and relationships based on a detailed manual analysis of the tools and how they support their features for asset management. When we observed conflicts between observations in multiple tools, we chose a higher-level concept that encapsulates them, in line with one of the main intentions of meta-modeling—reducing information complexity by abstraction [110]. We carried out the domain modeling in three phases: i) One author performed the initial design of the metamodel to establish its classes and their relationships in a single tool. ii) We adopted an iterative process to refine the class relationships from the initial design while considering other tools. This involved weekly meetings with all authors, where we reviewed and iteratively improved the metamodel design until all authors approved the metamodel. iii) We performed a validation phase where we populated our metamodel with concrete experiment information from actual experimental revision histories to reveal design flaws and identify improvement opportunities.

Results.

Figure 1.15 shows EMMM, a metamodel unifying the asset types from seventeen tools and their relationships. EMMM is a ready-to-use software artifact, formalized in Ecore, directly usable to facilitate tool development. Via the EMF-generated code [124] that we provide with the metamodel, it provides APIs and standard editors for manipulating its instances. We briefly describe the concepts represented in the metamodel below.

The metamodel presents class `Experiment` as the top-level asset of the metamodel and references multiple instances of the class `Metadata` to represent such information such tags, requirements, and authors. *Experiment Run* represents the asset type associated with other asset types used in a particular run. EMMM supports bookkeeping the exact versions of the assets used in such runs using the class `Run` with the attribute `versionId`. The class references `Metadata` to support storing additional metadata. The class `Run` is related to the following experiment assets through the abstract class `Asset`: i) Datasets and features, ii) Implementation assets and their parameters, iii) Execution results and performance data from a run. A specific version of an asset can be shared by multiple instances of `Run`, or it can be unique to an instance of `Run`.

`Asset` is an abstract class modeling various concrete asset types used during experiments. The attribute `versionId` tracks the different versions of assets created as an experiment evolves. The snapshot version of assets inherited from the abstract class has a one-to-one or one-to-many relationship with an instance of `Run`. Following the `Asset Types` established in *Contribution 2*

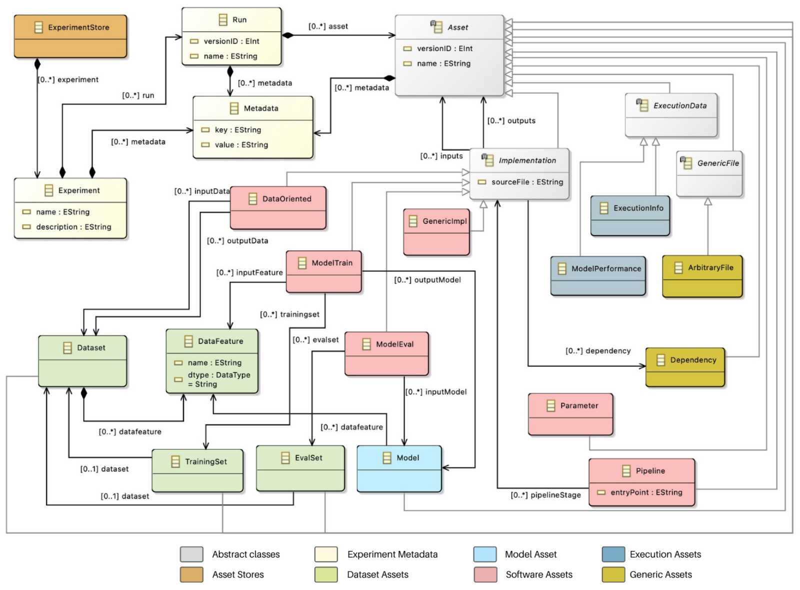

Figure 1.15: *EMMM*: Metamodel unifying all asset types and their relationships extracted from the 17 subject tools.

(Section 1.4.2), we represent the following observed asset types: `resource`, `software`, `metadata`, and `execution` assets.

Resource: The *Resource* assets include dataset, models, dependency, and generic assets. We introduce the class `Dataset` as a kind of `Asset` that represents the input data available for an experiment. The class `Dataset` contains multiple `DataFeature`, which store features in a specific dataset. During an experiment, data transformations or modifications create new instances of `Dataset` while incrementing the value of its inherited `versionId` attribute. Irrespective of the state of `Dataset`, the instances of `DataFeature` used during a particular run may be a subset of or all the features contained in the latest `Dataset` instance. This justifies the need for a separate representation of data features as class `DataFeature` and their type.

Some subject tools support the storage and tracking of `Model` as an asset type to support model-specific operations, such as model comparison across different runs. The subclass `Model`, by inheriting from `Asset`, can store model-related metadata. The class `Model` also references `DataFeature` to store information on the features or schema that the model supports. Assets connected to a model can be retrieved through the class `Run`, which contains all its associated assets. We include class `Dependency`, referenced by `Implementation` assets, to store all dependency or environment-related files and metadata. Our metamodel has a concrete class `ArbitraryFile`, which inherits the abstract class `GenericFile`—a type of `Asset`— to store arbitrary files. The abstract class `GenericFile` must be extended to store other custom files.

Software assets: Implementation related assets are supported through the classes `DataOriented`, `ModelTrain`, `ModelEval`, `GenericImpl`, `Parameter`, and `Pipeline`. We include the class `Implementation`, pointing to a `sourceFile` taking other `Assets` as inputs or outputs. The classes `DataOriented`, `ModelTrain`, and `ModelEval` are `Implementation` types based on the aspect of the ML workflow they represent. Since these classes inherit from abstract class `Asset`, the classes can store source code-related metadata. Classes `DataOriented`, `ModelTrain`, and `ModelEval` vary based on specific asset inputs and outputs they support. Class `DataOriented` represents implementation instances for preprocessing, transformation, or engineering of datasets, and it references class `Dataset` as input and output. The implementation of the training stage, where ML algorithms use training datasets to generate a new model, is represented by the class `ModelTrain`. Implementation of the performance evaluation of a model is represented by the class `ModelEval`. The class `GenericImpl` represents other implementations such as model deployment or monitoring that are not represented by `DataOriented`, `ModelTrain`, and `ModelEval`.

ExecutionData: The abstract class `ExecutionData` represents execution-related information that the subject tools track explicitly or automatically when executing experiment processes. It inherits from `Asset` and can be referenced as output generated by the class `Implementation`. These include 1) *ExecutionInfo*: which stores execution information generally tracked by the tools, e.g., terminal outputs, logs, bookkeeping information, and live hardware consumption. 2) *ModelPerfomance*, which stores the output of model evaluations. This is based on evaluation metrics, as tracked in different forms based on the ML task (e.g., sensitivity or ROC values for classification tasks; MSE, MAPE, or R^2 for regression tasks). Our metamodel stores model performance using class `ModelPerformance`, a subtype of `ExecutionData`.

Experiment Stores: Class `ExperimentStore` represents the storage of all experiment-related information and assets. How subjects physically store the actual `Experiment` information differs: some use file systems, others use databases, clouds, or a combination of those. The storage of assets may differ based on the asset types. For instance, the class `Dataset` can be stored in separate storage specific to data. In contrast, the class `Model` can be stored in another storage specific to models and their relevant metadata. Similarly, the class `GenericFile` can also be stored separately.

RQ3.2: Unified experiment management meta-model

Using domain analysis, we proposed EMMM, a unified metamodel representing the asset types, their relationships, and their evolution history among ML experiments as observed in 17 ML ExMTs. We propose EMMM as a reference for tool developers and researchers seeking to improve existing tools or develop next-generation tools with native support for ML. EMMM presents a superset of conceptualized structures and their relationships extracted from our subject tools. Our metamodel can foster the improvement of tools and the development of new tools with native support for ML assets.

Discussion.

Use cases: EMMM can be utilized in different forms. EMMM can be used to enable interoperability. Lack of interoperability is a weakness of existing ExMTs [125]. Our metamodel provides a foundation for enabling interoperability, offering an empirically informed representation of concepts from 17 tools. Developers of such tools can write import and export functions towards our metamodel; instead of one importer and exporter for each of the other tools, which might be costly. EMMM can be used as a blueprint for developing new tools. Extending available versioning tools such as Git towards native support for ML requires conceptualizing ML projects. Our metamodel provides such a conceptualization. Developers of tool extensions could represent the ML-specific information of a revision history as instances of our metamodel.

Configurable Tools: The variety of existing tools [78] can be ascribed to different user needs and scenarios. Even though we have presented a unified metamodel, not all valid uses require the support of the metamodel in its entirety. Instead, new tools might be desirable to implement support for a subset of the metamodel based on their specific needs. This leads to the notion of a *configurable* metamodel, in which a configuration can be described as views representing subsets of all the concepts and their relationships. So, configuring our metamodel with views on relevant assets can serve tools that require a subset of the metamodel. A configurable meta-data also provides opportunities for new kinds of tools. For example, tools with views on `DatasetFeature` can be used to trace model features back to concrete concepts within the application domain. Such information can be valuable knowledge for domain experts.

1.5 Summary of Publications

. This section presents the summary of publications appended to this thesis. The complete versions of the papers are in respective chapters following this introduction. We reformatted the papers to comply with the layout of this thesis.

Paper A

Machine Learning in District Heating System Energy Optimization

S. Idowu, S. Saguna, C. Ahlund, and O. Schelen

IEEE International Conference on Pervasive Computing and Communication Workshops, pp. 224-227. 2014

This paper presents a work in progress on applying ML to optimize energy in a district heating system. Specifically, the paper proposed using reinforcement learning and supervised ML on data collected from a district heating network. The reinforcement learning method solves the control-optimization problems, while the supervised learning method forecasts the heat load prediction as input for the former. The complex nature of district heating systems, including significant time delays and heat dissipation due to various factors, make ML an ideal optimization solution, though non-trivial. The proposed work involves acquiring relevant domain knowledge to facilitate decisions on data collection. For instance, *What are the essential domain data features to consider? What are the appropriate data sampling rates?* The paper proposes how ML methods can enhance energy efficiency in district heating substations. The paper identified and discussed a potential energy-saving strategy for district heating systems based on acquired domain knowledge. The paper argues for the use of online supervised machine learning methods to address data drift caused by changing underlying factors that the initial training data from the heating substations may not capture. For the proposed energy-saving strategy, the paper proposed using heat accumulators in buildings to reduce the peak demand on the central heating plants. Based on energy demand projection, the accumulators can store heat energy locally and discharge it for use during high demand in buildings. Online supervised learning methods forecast heat demand at each substation, while reinforcement learning methods control the charge/discharge of heat accumulators.

Paper B

Applied Machine Learning: Forecasting Heat Load in District Heating System

S. Idowu, S. Saguna, C. Ahlund, and O. Schelen

Elsevier Energy and Buildings, 133, pp. 478-488. 2016

This paper presents our efforts to optimize energy production, distribution, and consumption in district heating systems, building upon `Paper A`. Specifically, this paper presents an ML approach to forecast space and domestic water heating energy consumption in buildings. The work performed multiple machine-learning experiments and model prototyping based on acquired domain knowledge of district heating systems. The experiments are based on data collected unintrusively from ten residential and commercial buildings connected to our subject district heating system in Skellefteå, Sweden. We experiment with various supervised learning methods: support vector machine, regression tree, feed-forward neural network, and multiple linear regression. In addition, the work considers external factors such as weather conditions and internal factors such as physical parameters of heating substations as model input. The experiment and evaluation iterations use various learning methods and domain variables to identify the best-performing supervised learning methods and important domain features to forecast heat load for up to a 48-hour horizon effectively. In addition, we propose using a custom data feature engineering, which transforms the sequential data with lagging variables to account for the temporal relationship between the variables. This approach makes it possible to directly apply the considered classical learning methods to sequential data. The range of heat load forecast accuracy in substations obtained in this work is comparable with those obtained in related work that uses sequential-data-specific methods such as recurrent neural networks and adaptive time-series models. The paper found support vector machine and feed-forward neural network as the most efficient learning method, with regression tree as the worst performing method, reinforcing the reason behind the widespread use of such methods in related work. The paper also found that the internal factors of district heating systems are not significant domain features in forecasting energy consumption in buildings.

Paper C

On the Effectiveness of Machine Learning Experiment Management Tools

S. Idowu, O. Osman, D. Strüber, T. Berger

IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (SEIP), pp. 207-208. 2022

This short paper highlights the importance of managing artifacts (i.e., assets) when developing ML projects and ML-enabled systems. The paper proposes new research to explore the advancements and integration of a new class of tools—Experiment management tools—into established traditional SE tools. Specifically, this paper presents a research agenda and early results toward unified and practical SE and ML experiment management tools. The following summarizes the proposed research steps: (i) Assess usability and effectiveness. We propose using mixed-empirical methods, such as controlled experiments, surveys, and questionnaires, to elicit empirical data on the tool landscape. We also propose using controlled experiments to evaluate the effect of tools on user performance. (ii) Compare with SE tools. Here, studies should determine commonalities and differences across ExMTs and their equivalent traditional development tools. We advocate for the use of feature-based surveys relying on domain analysis techniques for such comparisons. (iii) Design and prototype unified tools. Relying on the results and outcome of the previous steps, we propose using meta-models in designing and presenting the conceptual structures in a unified form. (iv) Evaluate unified tools. The resulting tool prototypes should be evaluated with studies including controlled experiments, simulations, and action research of surveys.

Paper D

Asset Management in Machine Learning: State-of-research and State-of-practice

S. Idowu, D. Strüber, T. Berger

ACM Computing Surveys (CSUR), 55(7): 144:1-144:35. 2022

Based on the development challenges of ML-based software systems, this paper positions machine-learning asset management as an essential discipline that can provide improved methods and tools for operations on ML assets. This paper also presents a feature-based survey to understand the support that existing tools offer to facilitate research and practice on building new and improved management tools with native supports for ML and SE assets. Specifically, the paper answers the following research questions: *What ML assets are tracked and managed by state-of-research and state-of-practice tools? What are the mechanisms offered for collecting the assets? How are the assets stored and version-controlled? What are the management operations offered to users by the tools? What are the commonalities and variations between the state-of-research tools and the state-of-practice tools?* To answer the research questions, we present an overview of the features for managing assets used in ML experiments through a systematic survey of 18 state-of-practice and 12 state-of-research ExMTs. We identified their commonalities and variabilities and reported our findings using feature models. We reported four top-level features characterizing the tools as supported `Asset types`, `Collection` methods, `Storage` methods, and supported `Operations`. The sub-features of asset types as observed from our subjects are `Resource` (datasets, models, and arbitrary files), `Software` assets (source code, notebook, and parameters), `Metadata` (experiment, data, code, and model metadata), and `Execution asset` (dependency, jobs, execution metadata, and results). Our study shows that the state-of-practice and state-of-research tools support different asset types, predominantly metadata information describing the experiment, generic files, parameters, and results from experiment executions. Our subject's common asset collection methods are intrusive and require instrumentation in source code. We found that more than half of the state-of-practice tools delegate the storage of assets to third-party tools, and commonly supported operations include *tracking*, *exploring*, and *retrieving* assets aimed at experiment reproducibility.

Paper E

Machine Learning Experiment Management Tools: A Mixed-Method Empirical Study

S. Idowu, O. Osman, D. Strüber, T. Berger

Empirical Software Engineering (EMSE), 2023, [Under minor revision]

This paper investigates ML ExMTs and their support for users in a mixed-method empirical study. First, in a survey with 81 ML practitioners, we sought to understand the benefits and challenges of ML experiment management and the existing tool landscape. Second, we investigate the effectiveness of ML ExMTs in a controlled experiment with 15 student developers with a SE background. The paper addressed the following research questions: What kinds of experiments are conducted, and what ExMTs and features are used? What are the perceived benefits of using ExMTs? What are the challenges and adoption barriers of ExMTs? How does the adoption of ML ExMTs affect user performance? How are ML ExMTs, features, and paradigms perceived by users? The survey addressed the first three research questions, while the controlled experiment addressed the last two. Our survey results show that 80% of the respondent practitioners perform manual supervised ML experiments using a wide range of ExMTs since they reduced error and increased completion rates. 52% of our survey participants who do not use specialized management tools are unaware of ExMTs or their benefits. Due to a lack of knowledge, experience, or preference for custom solutions, others do not use ExMTs. The challenges and limitations experienced with such tools range from technical issues such as instability, vendor lock-in restriction, computing resource limitation, and high cost for SaaS-based tools to a steep learning curve. Our controlled experiment established that ExMTs improve user performance compared to ad hoc approaches to managing ML assets. When comparing the asset tracking mode from new users, we did not observe an unequivocal preference for either of our considered alternatives, i.e., the API-based instrumenting source code versus a non-intrusive CLI-based approach. However, survey respondents (practitioners) have a clear preference for the API-based ExMTs. For querying and retrieving assets from the management tools, our results from both groups show a preference for a GUI-based tool over a CLI-based ExMTs.

Paper F

A Large-Scale Study of ML-Related Python Projects

S. Idowu, Y. Sens, T. Berger, M. Vierhauser, and J. Kruger

2024, [Under review]

The development of ML-enabled software systems is becoming increasingly complex, and the standard toolchains for managing these projects are still immature. This is primarily due to a lack of comprehensive understanding of the properties and challenges associated with ML-enabled software projects. This paper extensively studies over 31,066 ML projects hosted on GitHub, focusing on Python and popular ML libraries—TensorFlow and SciKit-learn. In this study, we aim to address three key research questions: i) What types of ML-related projects are maintained on GitHub? ii) Which development stages are commonly found in these ML-related projects? iii) How do these ML projects evolve over time? We reveal that most ML-related projects hosted on GitHub are geared toward research and education, indicating that ML technology is not yet widely integrated into major open-source systems. The study also shows that data-related development stages dominate these projects and are the most frequently updated ML asset types. Interestingly, the model evaluation stage is the least found activity in our subject projects, followed by the model training and prediction stages. The evolution pattern observed across all project commits indicates a consistent level of development activities for most ML development stages. Unlike the assumptions based on the general ML workflow, which indicate that data-related stages precede the model-related stages, this study shows that data preparation stage activities increase while data acquisition decreases in more projects as they evolve, offering insights into the aspects of ML development that tool developers and researchers should prioritize. This paper contributes a large-scale dataset and empirical data that provide valuable insights into the types of projects, their properties, and how they evolve. These findings are particularly useful for researchers, practitioners, and tool developers aiming to improve SE practices for ML-enabled projects.

Paper G

Unified Meta-Model for Tracking Machine Learning Experiments

S. Idowu, D. Strüber, T. Berger

Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2022

The current alternatives for improving the management of ML assets include i) Adopting dedicated ML ExMTs, which are gaining popularity for supporting concerns such as versioning, traceability, auditability, collaboration, and reproducibility; ii) Developing new and improved version control tools with support for domain-specific operations tailored towards ML assets. This paper contributes toward both improvement options: improving existing ML ExMTs and developing next-generation versioning tools. We present the Experiment Management Meta-Model (EMMM), a meta-model that unifies concepts and relationships extracted from systematically selected ExMTs, focused on the concept of experimental runs. Our meta-model characterizes two main concerns: (i) ML asset structures as concepts and their relationship as observed in the state-of-the-art tools; (ii) conceptual version control structures that can hold ML and traditional assets. We explained the meta-model's concepts and relationships and proposed it based on the Eclipse Modeling Framework (EMF) with its meta-modeling language, Ecore, to encode model structures. We evaluate our meta-model on a real ML experiment scenario, validating its usefulness and suitability for capturing actual revision histories of ML projects. In addition, we discuss the improvements enabled by our meta-model in terms of possible use cases. Our proposed model can be used as a blueprint for practitioners to improve existing tools (targeting data scientists) and for researchers to develop new tools (targeting software engineers) with capabilities to support the identified concepts and relationships natively.

1.6 Conclusion

With the rise of ML-enabled software systems, modern systems have become more diverse. Instead of traditional software code artifacts, ML-related assets like datasets, features, and models are now additionally used, which has introduced new challenges in the development and production of ML-enabled systems. Traditional SE tools are no longer effective in managing the extended asset types introduced by ML components and the non-deterministic nature of ML. To tackle these challenges, a new class of tools called ExMTs has been developed. These tools provide practitioners with ways to track and retrieve development assets, ensuring reproducibility, traceability, and collaboration. However, these tools are not yet fully matured, have limited integration with SE tools, and are not primarily designed for software engineer users.

This thesis aims to facilitate next-generation management tools that natively support ML and SE assets and their management operations, integrate with existing SE tools, and aid software engineers. Through knowledge-seeking and solution-seeking research, we present our results, which researchers and tool developers can use to facilitate and guide the development of new and improved tools: i) Through an exploratory study and practitioner survey, we present the challenges of effectively managing ML experiments without specialized management tools. ii) We report insights on ML experiment projects' types and development stages. Our results show that most ML projects stored in GitHub are geared toward research and education, and most development activity centers around data processing. iii) We present the landscape overview of state-of-practice and state-of-research ExMTs and their features, including supported asset types, the collection, storage method, and the supported operations. The primary asset type classes they support are resources (data-related assets & models), software (source code), metadata, and execution data(pipelines, metrics, and execution logs). The collection methods are either intrusive or non-intrusive, while assets are stored locally or externally (e.g., cloud storage) in file systems, databases, and repositories. Their primary operations allow users to log or track their assets' versions and explore the stored assets for experiment concerns such as analysis, traceability, reproducibility, management, and collaboration. We found tools used in practice have extended features covering different asset management classes, implying that multi-purposed tools are commonly desired in practice. iv) We report practitioners' perceived benefits and challenges of the existing ML ExMTs. We found that practitioners find ExMTs highly beneficial as they facilitate their ML experiment tasks and help them perform experiments efficiently. They also experience vendor lock-in, computing resource limitations, costs, steep learning curves, poor documentation, bugs, and poor collaboration features as limitations or challenges. v) We present evidence of the effectiveness of ML ExMTs on user performance. Our result established that ML experiment tools effectively improve user performance during ML experiment development. vi) Following the empirical results from the previous contributions, we propose a research path to achieve integrated and practical SE tools and ExMTs. vii) Lastly, we propose a blueprint of unified concepts and structure from multiple ML ExMTs to facilitate the development of next-generation ExMTs.

For future work, we identify two primary directions: i) further empirical investigations into ML asset management tools and ii) prototyping tools based on findings of practical tools. Further empirical research is needed to examine the commonalities and differences between traditional SE tools (e.g., VCS) and ML ExMTs. We recommend such studies to investigate the supported assets, operations, collection modes, interfaces, and user-interaction modes. Similarly, further research can explore other asset management classes beyond ML experiment management. We recommend new studies on conducting controlled large-scale experiments with practitioners working on different categories of real-world ML tasks.

Further research is also needed to act on our empirical findings by building prototype tools and using them as guides on the areas for improvement and features for new tools. Based on the findings of this thesis, we identify the following as critical areas for improvement. We recommend new tools be configurable tools. This thesis establishes that many ML ExMTs exist, and ML projects have heterogeneous structures due to differing development purposes. It follows that tools should not assume users' workflow or restrict them on how to manage assets. Instead, tools should offer configurable options. For example, a common tool with several add-ons can be activated or deactivated based on the user's preference or development use case. Similarly, tools should support multiple paradigms (i.e., multimodal) for asset collection and user interaction. For example, the latest version of DVC (a formally CLI-based tool) now supports Web UI. Also, new tools should strive for automatic asset collection with low intrusion to reduce error or usage costs. Beyond the support for reproducibility, which is the current focus of many ExMTs, new tools should provide extended support for ML-specific use cases in exploring and managing assets. Lastly, new tools should integrate experiment management features into software development IDEs. Such integration has the potential to improve user adoption, experience, and usage. Lastly, future research should improve interoperability across existing tools, for instance, our *EMMM* tools can be used to support importing and exporting experiment assets from existing tools provided by different vendors.

Bibliography

[1] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

[2] T. Xie, "Intelligent Software Engineering: Synergy Between AI and Software Engineering," in *Dependable Software Engineering. Theories, Tools, and Applications*, X. Feng, M. Müller-Olm, and Z. Yang, Eds. Cham: Springer International Publishing, 2018, pp. 3–7.

[3] E. d. S. Nascimento, I. Ahmed, E. Oliveira, M. P. Palheta, I. Steinmacher, and T. Conte, "Understanding development process of machine learning systems: Challenges and solutions," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–6.

[4] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 50–59.

[5] F. Ishikawa and N. Yoshioka, "How do engineers perceive difficulties in engineering of machine-learning systems? - questionnaire survey," in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2019, pp. 2–9.

[6] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *SEAA*. IEEE, 2018, pp. 50–59.

[7] Y. Dang, Q. Lin, and P. Huang, "Aiops: Real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 4–5.

[8] F. Kumeno, "Sofware engineering challenges for machine learning applications: A literature review," *Intelligent Decision Technologies*, vol. 13, no. 4, pp. 463–476, 2020.

[9] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International*

Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2019, pp. 291–300.

[10] R. Ranawana and A. S. Karunananda, "An agile software development life cycle model for machine learning application development," in *2021 5th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI).* IEEE, 2021, pp. 1–6.

[11] H. Kuwajima, H. Yasuoka, and T. Nakae, "Engineering problems in machine learning systems," *Machine Learning*, vol. 109, no. 5, pp. 1103–1126, 2020. [Online]. Available: https://doi.org/10.1007/s10994-020-05872-w

[12] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, vol. 180, p. 111031, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100128X

[13] M. A. Al Alamin and G. Uddin, "Quality assurance challenges for machine learning software applications during software development life cycle phases," in *2021 IEEE International Conference on Autonomous Systems (ICAS)*, 2021, pp. 1–5.

[14] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine learning: The high interest credit card of technical debt," 2014.

[15] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2503–2511. [Online]. Available: http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

[16] J. Bosch, H. H. Olsson, B. Brinne, and I. Crnkovic, "Ai engineering: Realizing the potential of ai," *IEEE Software*, vol. 39, no. 6, pp. 23–27, 2022.

[17] J. Bosch, "Introduction to the ai engineering theme," *Accelerating Digital Transformation: 10 Years of Software Center*, p. 399, 2022.

[18] F. Khomh, B. Adams, J. Cheng, M. Fokaefs, and G. Antoniol, "Software engineering for machine-learning applications: The road ahead," *IEEE Software*, vol. 35, no. 5, pp. 81–84, 2018.

[19] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 162–170.

[20] M. Vartak, H. Subramanyam, W.-E. E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "ModelDB: a system for machine learning model management," in *the Workshop.* ACM Press, aug 2016, pp. 1–3.

[21] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert, "Declarative Metadata Management: A Missing Piece in End-To-End Machine Learning," *SysML 2018*, p. 3, 2018.

[22] A. Polyzotis, M. A. Zinkevich, S. Whang, and S. Roy, "Data Management Challenges in Production Machine Learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1723–1726.

[23] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *International Conference on Agile Software Development.* Springer, Cham, 2019, pp. 227–243.

[24] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, vol. 180, p. 111031, 2021.

[25] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. Roselli, and N. Talagala, "Model governance: Reducing the anarchy of production ML," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 351–358.

[26] R. Tatman, J. Vanderplas, and S. Dane, "A Practical Taxonomy of Reproducibility for Machine Learning Research," *Reproducibility in ML Workshop, ICML'18*, no. Ml, 2018.

[27] R. Isdahl and O. E. Gundersen, "Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms," in *eScience.* IEEE, 2019. [Online]. Available: https://dx.doi.org/10.1109/eScience.2019.00017

[28] A. L. Beam, A. K. Manrai, and M. Ghassemi, "Challenges to the reproducibility of machine learning models in health care," *Jama*, vol. 323, no. 4, pp. 305–306, 2020.

[29] C. Drummond, "Replicability is not reproducibility: nor is it good science," 2009.

[30] R. R. Bouckaert, "Estimating replicability of classifier learning experiments," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 15.

[31] M. Mora-Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, and M.-A. Sicilia, "Traceability for trustworthy ai: A review of models and tools," *Big Data and Cognitive Computing*, vol. 5, no. 2, p. 20, 2021.

[32] M. Schlegel and K.-U. Sattler, "Management of machine learning lifecycle artifacts: A survey," *arXiv preprint arXiv:2210.11831*, 2022.

[33] S. Idowu, D. Strüber, and T. Berger, "Asset management in machine learning: State-of-research and state-of-practice," *ACM Computing Surveys (CSUR)*, 2022.

[34] I. H. Sarker, F. Faruque, U. Hossen, and A. Rahman, "A Survey of Software Development Process Models in Software Engineering," *International Journal of Software Engineering and Its Applications*, vol. 9, no. 11, pp. 55–70, 2015. [Online]. Available: http://10.0.55.177/ijseia.2015.9.11.05https://dx.doi.org/10.14257/ijseia.2015.9.11.05

[35] R. Wirth, "CRISP-DM : Towards a Standard Process Model for Data Mining," *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, vol. 1, no. 24959, pp. 29–39, 2000.

[36] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD Process for Extracting Useful Knowledge from Volumes of Data," *Commun. ACM*, vol. 39, no. 11, pp. 27–34, 1996.

[37] Microsoft, "Team Data Science Process Documentation," 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/

[38] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. Parameswaran, "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*, ser. DEEM'18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3209889.3209897

[39] X. Bouthillier and G. Varoquaux, "Survey of machine-learning experimental methods at neurips2019 and iclr2020," Tech. Rep., 2020.

[40] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine*, vol. 104, p. 101822, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0933365719310437

[41] L. Tuggener, M. Amirian, K. Rombach, S. Lörwald, A. Varlet, C. Westermann, and T. Stadelmann, "Automated machine learning in practice: State of the art and recent results," in *2019 6th Swiss Conference on Data Science (SDS)*, 2019, pp. 31–36.

[42] D. Zhang, Y. Shen, Z. Huang, and X. Xie, "Auto machine learning-based modelling and prediction of excavation-induced tunnel displacement," *Journal of Rock Mechanics and Geotechnical Engineering*, vol. 14, no. 4, pp. 1100–1114, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1674775522000786

[43] H. H. Rashidi, N. Tran, S. Albahra, and L. T. Dang, "Machine learning in health care and laboratory medicine: General overview of supervised learning and Auto-ML," *International Journal of Laboratory Hematology*, vol. 43, no. S1, pp. 15–22, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/ijlh.13537

[44] ISO/IEC/IEEE, "IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999) IEEE Standard for Information Technology—System and Software Life

Cycle Processes—Reuse Processes," *IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999)*, pp. 1–51, 2010.

[45] S. Gollapudi, *Practical Machine Learning*, ser. Community experience distilled. Packt Publishing, 2016.

[46] D. N. da Silva, A. Simões, C. Cardoso, D. E. de Oliveira, J. N. Rittmeyer, K. Wehmuth, H. Lustosa, R. S. Pereira, Y. Souto, L. E. Vignoli, R. Salles, S. C. de Heleno, A. Ziviani, E. Ogasawara, F. C. Delicato, P. F. de Pires, H. L. C. da Pinto, L. Maia, and F. Porto, "A conceptual vision toward the management of machine learning models," in *CEUR Workshop Proceedings*, vol. 2469, 2019, pp. 15–27. [Online]. Available: http://www.master.iag.usp.br/lab/

[47] "Azure ai — microsoft cloud," 2022. [Online]. Available: https://azure.microsoft.com/

[48] G. Berg, "Image classification with machine learning as a service:-a comparison between azure, sagemaker, and vertex ai," 2022.

[49] "Amazon SageMaker." [Online]. Available: https://aws.amazon.com/sagemaker/

[50] "Vertex ai — google cloud," 2022. [Online]. Available: https://cloud.google.com/vertex-ai

[51] "Datarobot ai — microsoft cloud," 2022. [Online]. Available: https://www.datarobot.com/platform/

[52] L. Projects, "Mlflow," https://mlflow.org/, 2021.

[53] "Polyaxon - machine learning at scale." [Online]. Available: https://polyaxon.com/

[54] D. V. Control, "What is dvc?" [Online]. Available: https://dvc.org/doc/user-guide/what-is-dvc

[55] A. A. Ormenisan, M. Ismail, S. Haridi, and J. Dowling, "Implicit Provenance for Machine Learning Artifacts," *MLSys'20*, p. 3, 2020.

[56] R. Garcia, V. Sreekanti, N. Yadwadkar, and Others, "Context: The missing piece in the machine learning lifecycle," *KDD CMI*, 2018. [Online]. Available: https://rlnsanz.github.io/dat/Flor{_}CMI{_}18{_}CameraReady.pdf

[57] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, and Y. Wu, "Vamsa: Tracking provenance in data science scripts," 2020.

[58] J. Schad, R. Sambasivan, and C. Woodward, "Arangopipe, a tool for machine learning meta-data management," *Data Science*, no. Preprint, pp. 1–15, 2021.

[59] M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthör, K. Innerebner, F. Klezin, S. Lindstaedt, A. Phani, B. Rath *et al.*, "Systemds: A declarative machine learning system for the end-to-end data science lifecycle," *arXiv preprint arXiv:1909.02976*, 2019.

[60] S. Shankar and A. Parameswaran, "Towards observability for machine learning pipelines," *arXiv preprint arXiv:2108.13557*, 2021.

[61] H. Miao, A. Chavan, and A. Deshpande, "ProvDB: Lifecycle management of collaborative analysis workflows," *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics, HILDA 2017*, no. d, 2017.

[62] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "ModelHub : Lifecycle Management for Deep Learning," *University of Maryland*, vol. 0, 2016.

[63] G. Gharibi, V. Walunj, S. Rella, and Y. Lee, "ModelKB: Towards automated management of the modeling lifecycle in deep learning," *Proceedings - 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2019*, no. March, pp. 28–34, 2019.

[64] C. Weber, P. Hirmer, and P. Reimann, "A Model Management Platform for Industry 4.0 – Enabling Management of Machine Learning Models in Manufacturing Environments," in *Business Information Systems.* Springer International Publishing, 2020, pp. 403–417. [Online]. Available: http://10.0.3.239/978-3-030-53337-3{_}30https://dx.doi.org/10.1007/978-3-030-53337-3{_}30

[65] M. Vartak, J. M. Da Trindade, S. Madden, and M. Zaharia, "MISTIQUE: A system to store and query model intermediates for model diagnosis," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1285–1300, 2018.

[66] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A {Low-Latency} online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 613–627.

[67] "Bento ml cloud," 2022. [Online]. Available: https://www.bentoml.com/

[68] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving," *arXiv preprint arXiv:1712.06139*, 2017.

[69] "Cloud infrastructure for machine learning at scale," 2022. [Online]. Available: https://www.cortex.dev/

[70] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, V. Shah, B. Shen, L. Sugden, K. Zhao, and M.-C. a. Wu, "Data Platform for Machine Learning," in *Proceedings of the 2019 International Conference on Management of Data.* ACM, 2019. [Online]. Available: http://10.0.4.121/3299869.3314050https://dx.doi.org/10.1145/3299869.3314050

[71] S. Baunsgaard, M. Boehm, A. Chaudhary, B. Derakhshan, S. Geißelsöder, P. M. Grulich, M. Hildebrand, K. Innerebner, V. Markl, C. Neubauer *et al.*, "Exdra: Exploratory data science on federated raw data," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2450–2463.

[72] “Pachyderm — Version-controlled data science.” [Online]. Available: https://www.pachyderm.com/

[73] Neptune, “Neptune.ai,” https://neptune.ai/, 2021.

[74] “StudioML A Python model management framework.” [Online]. Available: https://www.studio.ml/

[75] “Weights & Biases – Developer tools for ML.” [Online]. Available: https://www.wandb.com/

[76] “Comet – Build better models faster!” 2022. [Online]. Available: https://www.comet.ml/site/

[77] L. Quaranta, F. Calefato, and F. Lanubile, “A taxonomy of tools for reproducible machine learning experiments,” 2021.

[78] S. Idowu, D. Strüber, and T. Berger, “Asset management in machine learning: A survey,” in *ICSE-SEIP*. IEEE, 2021, pp. 51–60.

[79] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning.* MIT press Massachusetts, USA:, 2017, vol. 1.

[80] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, “Data Lifecycle Challenges in Production Machine Learning: {A} Survey,” *{SIGMOD} Rec.*, vol. 47, no. 2, pp. 17–28, 2018.

[81] A. Kumar, M. Boehm, and J. Yang, “Data management in machine learning: Challenges, techniques, and systems,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1717–1722.

[82] R. Ferenc, T. Viszkok, T. Aladics, J. Jász, and P. Hegedűs, “Deep-water framework: The Swiss army knife of humans working with machine learning models,” *SoftwareX*, vol. 12, p. 100551, 2020. [Online]. Available: https://doi.org/10.1016/j.softx.2020.100551

[83] T. Weißgerber and M. Granitzer, “Mapping platforms into a new open science model for machine learning,” *it - Information Technology*, vol. 61, no. 4, pp. 197–208, 2019. [Online]. Available: https://dx.doi.org/10.1515/itit-2018-0022

[84] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, “Findings from github: methods, datasets and limitations,” in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 137–141.

[85] A. Bhatia, E. E. Eghan, M. Grichi, W. G. Cavanagh, Z. Ming, B. Adams *et al.*, “Towards a change taxonomy for machine learning systems,” *arXiv preprint arXiv:2203.11365*, 2022.

[86] D. Gonzalez, T. Zimmermann, and N. Nagappan, “The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github,” in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 431–442.

[87] B. van Oort, L. Cruz, M. Aniche, and A. van Deursen, "The prevalence of code smells in machine learning projects," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 1–8.

[88] A. J. Simmons, S. Barnett, J. Rivera-Villicana, A. Bajaj, and R. Vasa, "A large-scale comparative analysis of coding standard conformance in open-source data science projects," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–11.

[89] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[90] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, 2003.

[91] S. Idowu, S. Saguna, C. Ahlund, O. Schelen, C. Åhlund, and O. Schelén, "Applied machine learning: Forecasting heat load in district heating system," *Energy and Buildings*, vol. 133, pp. 478–488, dec 2016. [Online]. Available: http://dx.doi.org/10.1016/j.enbuild.2016.09.068

[92] L. Gao, X. Cui, J. Ni, W. Lei, T. Huang, C. Bai, and J. Yang, "Technologies in smart district heating system," *Energy Procedia*, vol. 142, pp. 1829–1834, 2017.

[93] H. Gadd and S. Werner, "Heat load patterns in district heating substations," *Applied energy*, vol. 108, pp. 176–183, 2013.

[94] W. Tellis, "Application of a case study methodology," *The qualitative report*, vol. 3, no. 3, pp. 1–19, 1997.

[95] S. Biswas, M. Wardat, and H. Rajan, "The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large," *arXiv preprint arXiv:2112.01590*, 2021.

[96] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[97] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471

[98] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis {(FODA)} Feasibility Study," Carnegie-Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1990.

[99] D. Nešić, J. Krüger, S. Stănciulescu, and T. Berger, "Principles of Feature Modeling," in *FSE*, 2019.

[100] A. J. Ko, T. D. LaToza, and M. M. Burnett, “A practical guide to controlled experiments of software engineering tools with human participants,” *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[101] A. Jedlitschka and D. Pfahl, “Reporting guidelines for controlled experiments in software engineering,” in *2005 International Symposium on Empirical Software Engineering, 2005.* IEEE, 2005, pp. 10–pp.

[102] S. Counsell, “Do student developers differ from industrial developers?” in *ITI*, 2008, pp. 477–482.

[103] I. Salman, A. T. Misirli, and N. Juristo, “Are students representatives of professionals in software engineering experiments?” in *ICSE*, vol. 1, 2015, pp. 666–676.

[104] M. Höst, B. Regnell, and C. Wohlin, “Using students as subjects—a comparative study of students and professionals in lead-time impact assessment,” *ESE*, vol. 5, no. 3, pp. 201–214, 2000.

[105] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert, and J. Siegmund, “Efficiency of projectional editing: A controlled experiment,” in *FSE*, 2016, p. 763–774.

[106] P. Runeson, “Using students as experiment subjects–an analysis on graduate and freshmen student data,” in *EASE*, 2003, pp. 95–102.

[107] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, “Empirical software engineering experts on the use of students and professionals in experiments,” *ESE*, pp. 452–489, 2018.

[108] R. Wieringa, “Design science methodology: principles and practice,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, 2010, pp. 493–494.

[109] R. J. Wieringa, *Design science methodology for information systems and software engineering.* Springer, 2014.

[110] A. Wasowski and T. Berger, *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*, 2022. [Online]. Available: http://dsl.design

[111] S. Biswas, M. J. Islam, Y. Huang, and H. Rajan, “Boa meets python: a boa dataset of data science software in python language,” in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* IEEE, 2019, pp. 577–581.

[112] A. Barrak, E. E. Eghan, and B. Adams, “On the co-evolution of ml pipelines and source code-empirical study of dvc projects,” in *IEEE International Conference on Software Analysis, Evolution and Reengineering.* IEEE, 2021, pp. 422–433.

[113] D. Ramasamy, C. Sarasua, A. Bacchelli, and A. Bernstein, “Workflow analysis of data science code in public github repositories,” *Empirical Software Engineering*, vol. 28, no. 1, pp. 1–47, 2023.

[114] Ml-Tooling, “Ml-tooling/best-of-ml-python: A ranked list of awesome machine learning python libraries. updated weekly.” [Online]. Available: https://github.com/ml-tooling/best-of-ml-python

[115] “Most popular machine learning libraries - 2014/2021,” Oct 2021. [Online]. Available: https://statisticsanddata.org/data/most-popular-machine-learning-libraries

[116] S. Raschka and V. Mirjalili, *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2.* Packt Publishing Ltd, 2019.

[117] A. R. Munappy, D. I. Mattos, J. Bosch, H. H. Olsson, and A. Dakkak, “From ad-hoc data analytics to dataops,” in *ICSSP*, 2020.

[118] A. R. Munappy, J. Bosch, and H. H. Olsson, “Data pipeline management in practice: Challenges and opportunities,” in *PROFES*, 2020.

[119] A. R. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, “Data management for production quality deep learning models: Challenges and solutions,” *Journal of Systems and Software*, vol. 191, p. 111359, 2022.

[120] H. Miao, A. Li, L. S. Davis, and A. Deshpande, “Towards Unified Data and Lifecycle Management for Deep Learning,” in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, pp. 571–582. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2017.112

[121] K.-J. Lui, “Sample size determination for a 3-treatment 3-period crossover trial in frequency data,” *Therapeutic innovation & regulatory science*, vol. 52, no. 4, pp. 407–415, 2018.

[122] J. R. Turner, *Crossover Design.* New York, NY: Springer New York, 2013, pp. 521–521. [Online]. Available: https://doi.org/10.1007/978-1-4419-1005-9_1009

[123] L. Linsbauer, F. Schwaegerl, T. Berger, and P. Gruenbacher, “Concepts of variation control systems,” *Journal of Systems and Software*, vol. 171, 2021.

[124] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework.* Pearson Education, 2008.

[125] M. Mora-Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, and M.-A. Sicilia, “Traceability for trustworthy ai: A review of models and tools,” *Big Data and Cognitive Computing*, vol. 5, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/2/20