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

SAMUEL IDOWU, Chalmers | University of Gothenburg, Sweden
DANIEL STRÜBER, Radboud University Nijmegen, Netherlands and Chalmers | University of Gothenburg, Sweden
THORSTEN BERGER, Chalmers | University of Gothenburg, Sweden and Ruhr University Bochum, Germany

Machine learning components are essential for today's software systems, causing a need to adapt traditional software engineering practices when developing machine-learning-based systems. This need is pronounced due to many development-related challenges of machine learning components such as asset, experiment, and dependency management. Recently, many asset management tools addressing these challenges have become available. It is essential to understand the support such tools offer to facilitate research and practice on building new management tools with native supports for machine learning and software engineering assets.

This article positions machine learning asset management as a discipline that provides improved methods and tools for performing operations on machine learning assets. We present a feature-based survey of 18 state-of-practice and 12 state-of-research tools supporting machine-learning asset management. We overview their features for managing the types of assets used in machine learning experiments. Most state-of-research tools focus on tracking, exploring, and retrieving assets to address development concerns such as reproducibility, while the state-of-practice tools also offer collaboration and workflow-execution-related operations. In addition, assets are primarily tracked intrusively from the source code through APIs and managed via web dashboards or command-line interfaces. We identify asynchronous collaboration and asset reusability as directions for new tools and techniques.

CCS Concepts: • **Software and its engineering**; • **Computing methodologies** → **Machine learning**;

Additional Key Words and Phrases: machine learning, experiment management tools, SE4AI

ACM Reference Format:
Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2022. Asset Management in Machine Learning: State-of-research and State-of-practice. 1, 1 (June 2022), 33 pages.

1 Introduction

The momentum behind machine learning is rapidly increasing as companies recognize it as a key enabling technology for today's and future business challenges [87]. Similar to how it is essential to manage traditional software engineering components during development, the effective management of machine learning components is vital for the success of machine-learning-based software systems. However, for several reasons, improving the effectiveness of developing such systems requires new, dedicated methods and tools.

Authors' addresses: Samuel Idowu, samuelid@chalmers.se, Chalmers | University of Gothenburg, Sweden; Daniel Strüber, Radboud University Nijmegen, Netherlands, Chalmers | University of Gothenburg, Sweden, d.strueber@cs.ru.nl; Thorsten Berger, Chalmers | University of Gothenburg, Sweden, Ruhr University Bochum, Germany, thorsten.berger@rub.de.

Manuscript submitted to ACM

First, developing machine-learning-based systems requires management of a greater variety of asset types than traditional software systems, including *resource artefacts* such as datasets, features, and models; *software artefacts* such as source code files and hyperparameters; and *metadata*, including experiment metadata, execution metadata and performance metrics [44]. Consequently, these assets require tools with relevant machine-learning-specific domain abstractions for effective management. For example, Amershi et al. report that Microsoft teams developing machine-learning-based systems found discovering, managing, and versioning machine-learning assets to be more complex and challenging than other types of software engineering [3].

Second, the machine learning workflow is a non-linear stage-by-stage process with feedback loops, requiring a certain amount of experimentation before converging to an acceptable model (Fig. 1). Similar to experimental workflows in scientific software [34] and hardware/software integrated designs [22], the machine learning workflow involves large sets of runs (a.k.a iterations) over different machine learning assets and quickly becomes complex. This already applies to non-production-focused scenarios (e.g., research projects and community contests), where it is common to explore tens to hundreds of runs [15, 15, 29, 54]. Production-focused scenarios may involve multiple machine learning components and are even more prone to feedback loops, making extended experimentation necessary [3, 13]. The explicit management of experiments and their associated assets can reduce the complexity and time overhead of managing assets in multiple runs [36, 71, 82]. Specifically, it can help model developers effectively explore experimentation history, avoid redundant effort, and recover previous experiment paths on demand. This holds just as much in the case of automatically configured runs (e.g., hyperparameter search), where developers might be interested in understanding the experimentation history for diagnostic purposes.

Third, in production-focused scenarios, machine learning assets such as trained models are integrated into software systems, in which their performance is continuously monitored [38]. A common activity is retraining models after more data has become available from newly encountered contexts or when data drift occurs. Such retraining activity then, again, requires a non-linear stage-by-stage process of experimentation with multiple runs (indicated in Fig. 1 with the upwards arrows from the DevOps stages). It is essential to have access to and understand the complete provenance of assets used to train the current and earlier models to make informed decisions on retraining models and debug in-production model behaviors. Such information can be instrumental in tracing model operational behavior to concrete experiment assets and actions, providing information on the experimental paths previously explored during earlier runs.

Traditional software engineering methods and tools, such as version control systems (VCS), have been designed with support for the management of software assets in mind. Therefore, they seem a tempting solution for the engineering of machine-learning-based software systems as well. However, for the issues mentioned above, several challenges are associated with directly adopting the traditional tools and methods for this purpose. For example, traditional VCSs such as Git do not support advanced, domain-specific queries on machine learning assets, e.g., *what data features and hyperparameter have been used in an experiment run in which the final model precision was 0.75 or greater?* That would be informative for model developers when understanding the history of a project. Generally, managing experimental run requires systematic provenance and versions of assets from current and previous runs to address different machine learning experiment concerns, including traceability [59], reproducibility [7, 46, 78], and auditability [68]. Due to the lack of explicit tooling, many practitioners adopt informal, ad hoc, or custom approaches to manage machine learning assets [36, 75]. For example, all the developers interviewed by Hill et al. [36] could not effectively version machine-learning-specific assets and, hence, used informal methods such as emails, spreadsheets, and notes to manage assets during machine learning model development. Such methods may suffice for small-scale experiments; however, large-scale experiments with many runs and collaboration requirements demand improved methods. Consequently, we

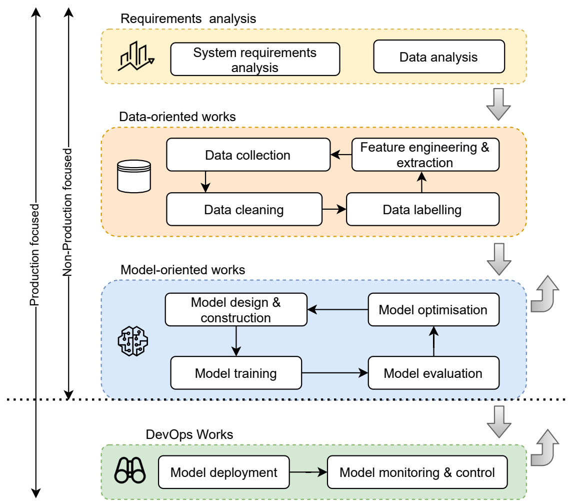

Fig. 1. Stages in a typical machine learning workflow

consider machine-learning asset management an essential discipline that offers improved machine-learning-specific asset management and operations to address developmental concerns when building machine-learning-based systems.

Machine-learning experiment management tools, an emerging class of asset management tools, aims to address the challenges of managing machine-learning-specific assets. Examples of such tools used in practice include *MLFlow*, *NeptuneML*, and *WandB*, while examples from the literature include *Deep-water* [27], *Runway* [79], and *ModelKB* [28, 29]. These tools target practical experiment concerns, including reproducibility [7, 46, 78] and traceability [59], by providing functionalities to store, track, and version assets from different experiment runs. While these tools have become available recently, they are not fully matured yet, especially compared to their traditional counterparts. Factors affecting their maturity include the lack of interoperability across different tools, tight coupling with specific libraries, friction, and overhead incurred during usage due to required code instrumentation for tracking assets [60, 64]. It is essential to assess the support found in the current tool landscape to facilitate research and practice towards improving existing tools, developing new ones, and improving the engineering processes of machine-learning-based systems. Important questions include: *What support do these tools provide to users? What are the asset types they track? What are their commonalities and variabilities?*

This paper discusses and positions asset management as an essential discipline that provides dedicated methods and tools to effectively address different concerns of machine learning experiments, especially for large-scale experiments. We survey asset management support found in state-of-research and state-of-practice, in existing and proposed management tools for machine learning experiments, identifying the types of assets supported and the operations offered to users for managing machine learning assets. Specifically, we conduct a feature-based survey—a domain analysis to identify the characteristics of an application domain (in our case, experiment-management tools). We model these characteristics as features in a feature model [48, 63], an intuitive tree-like notation commonly used in software variability management [4, 10]. In the literature, such feature-based surveys have been performed before to compare the design space of technologies, such as model transformations [20], conversational AI systems [5], language workbenches [25], and variation control systems [53].

Our study contributes a feature-model-based representation of tools supporting machine learning asset management—particularly experiment management tools—focusing on asset types and supported operations. We address research questions (see Section 3.1) about the asset types, collection, storage, and operations support offered by state-of-research and state-of-practice tools.

Our scope in terms of machine learning paradigms is primarily focused on supervised and unsupervised machine learning. This scope is a consequence of the support offered by the available tools identified via our systematic methodology (see Section 3), which generally provide support for those paradigms. We exclude from our scope tools that are only focused on one particular aspect of machine-learning asset management, e.g., model registries and model databases [57], tools for dataset and metadata management [81], pipeline orchestration, hyper-parameter management, and visualizations.

With our study, we contribute to an increased empirical understanding of the current solution space for machine-learning asset management. Practitioners can use our survey results to understand the asset management features provided by available tools. Researchers can also identify gaps in the tool support for asset management and classify their new techniques against our taxonomy (the feature model). Lastly, we hope that our result will contribute toward building new tools with improved asset management methods for developing machine-learning-based components.

An earlier version of this work appeared in a conference publication [44], where we defined and positioned asset management as a discipline to improve tools and techniques for engineering machine-learning-based systems and surveyed 17 state-of-practice experiment management tools. We have significantly extended this earlier paper in different ways. In particular, we consider 12 state-of-research tools as additional subject tools and extended the resulting feature model with new features. This extension also allows us to consider further research questions. In the previous work, we answered the research questions RQ1–RQ4 (see Section 3.1) for the state-of-practice tools. Now we also answer these research questions for the identified state-of-research tools. In addition, we now compare the state-of-practice and state-of-research tools by presenting a feature matrix across the subject tools.

We proceed by presenting relevant background in Section 2 and describing our methodology in Section 3. We present our results, including the feature model in Section 4, and discuss our findings in Section 5. In Section 6, we describe the threats to the validity of this work. We discuss related work in Section 7, future work in Section 8 and conclude in Section 9.

2 Machine Learning Asset Management

We now describe machine learning workflow, experiments, asset management challenges, and position machine-learning asset management.

2.1 Machine Learning Workflow & Machine Learning Experiments

The traditional software engineering process [69] includes requirements analysis, planning, architecture design, coding, testing, deployment, and maintenance stages. Similarly, supervised machine learning follows well-defined processes grounded in workflows designed in the data science and data mining context. Examples include CRISP-DM [84], KDD [26], and TDSP [56]. Figure 1 shows a simplified workflow of a supervised machine learning process lifecycle, structured along different development stages. The workflow consists of the stages for requirements analysis, data-oriented works, model-oriented works, and DevOps works [3, 6, 51]. The requirements analysis stages involve analyzing the system requirements and available data, while the data-oriented stages include data collection, cleaning, labeling, and feature engineering or extraction. Model-oriented stages include model design, training, evaluation, and optimization. The DevOps stages include deploying machine learning models and operationalizing —monitoring and controlling—in-production models. Figure 1 also shows that machine learning projects can either be production-focused or non-production-focused. For example, machine learning projects for research papers are often non-production focused and do not require DevOps operations. In contrast, machine-learning-based software projects are production-focused because they integrate and operationalize models.

The exploratory and experimentation-oriented nature of machine learning projects significantly differs from traditional software engineering. The workflow diagram in Fig. 1 contains a linear progression from requirements analysis to DevOps stages; however, machine learning workflows are typically non-linear and include multiple feedback loops (indicated by the upward arrows) [3]. These feedback loops reflect the multiple experiment *runs* (a.k.a *experiment iterations*) often performed during machine learning model development. A run refers to 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 task. The machine learning workflow relies on the multiple runs of trial-and-error steps due to the unpredictable nature of machine-learning model performance [3, 15, 88].

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 [6]. 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. Maintaining the provenance of the assets and processes used during these runs is essential to address important management concerns of machine learning model development. 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.

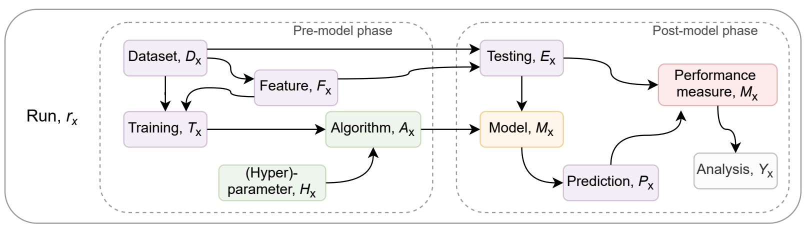

Fig. 2. Representation of a supervised machine-learning experiment run

Figure 2 illustrates different asset types that can be modified within the solution space of a specific run. 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 user-intuitive decisions on necessary step-by-step modifications for new runs. In contrast, the automatic approach systematically searches a pre-defined portion of the solution space (e.g., a set of hyper-parameters range) for each run—for example, optimization experimentation using training loops. Regardless of the employed approach, several experiment runs are often performed before arriving at the experiment's goal. 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 [36, 71, 82].

2.1.1 Challenges of Asset Management in Machine Learning Experiments The challenges encountered during machine learning experiments are often related to the lack of explicit tooling support to address experiment management concerns, including reproducibility [7, 46, 78], replicability [14, 24], traceability [59], explainability [16, 67], interpretability [17, 31], collaboration [89], and auditability [68]. In what follows, we conceptually describe different challenges of asset management for machine learning experiments.

Standardized management methods. There is a lack of standardized and explicit management methods to store, version, or operate the machine learning assets. Users rely on ad hoc approaches that may limit their efficiency during model development. For example, it may be difficult to reuse or compare assets, operations, and techniques across multiple projects [3]. The varying asset type formats (e.g., different language codebases, model, data, and metrics formats) contribute to this challenge and limit knowledge transfer regarding asset management across various projects.

Researchers recently started developing approaches for fostering interoperability and reuse of machine learning assets across multiple projects. For example, Mitchell et al. propose a generic framework intended to increase the transparency of models across different application contexts and stakeholders [58].

Tracking assets and operations. Tracking machine learning assets and operations, their versions, and the decisions engaged from one specific run to another are vital for effective machine learning asset management. Tracking the assets and their corresponding information serves as a foundation to support different machine learning concerns. For example, snapshots of assets, operations, and decisions should be captured per experiment run to enable support for traceability and audibility on factual questions such as: *what version of a particular asset was used for a specific experiment run?* or *what operations or thought processes were considered at a specific point of an experiment?*

Domain-specific operations. Asset management operations should be offered at the right abstraction level. Similar to how traditional software IDEs support representation and querying of code artifacts (e.g., functions, variables, and interfaces), machine learning asset management needs to offer domain-specific asset operations on abstraction levels specific to machine learning. For example, it should allow querying for data features or hyperparameters used in a specific run with model evaluation values as conditions. Such operations can be supported through dedicated artifact meta-models [37].

Managing experiment concerns. Acquiring effective methods to address experiment concerns such as reproducibility, replicability, and traceability has the potential to improve model development processes. However, there is a lack of adequate tools that explicitly support such concerns. For reproducibility, apart from dataset and code, essential experiment dependencies such as random seeds (for non-deterministic experiments) have to be systematically captured during the experiment to offer developers the possibility to reproduce experiments. Traceability is an important concern requiring systematic tracking and querying of machine-learning assets to trace operational models' behavior to concrete experiments.

Users and collaboration. Collaboration between multiple developers is a 'soft challenge' when working on machine learning experiments. This is partly due to the lack of suitable tools and workflows. Currently, collaborations between machine learning developers primarily involve sharing experiment resources or resulting models and performance metrics. We believe that better tools suited for specific machine learning assets have the potential to improve collaboration opportunities for developers. For example, versioning tools that support merging and branching for different asset types, including models, can foster teamwork and asynchronous or concurrent collaboration among developers.

Comparing, analyzing, and interpreting results. A great deal of machine learning result analysis requires the use of visualization for easy explanation and interpretation. Hence, the ability to effectively infer decisions and conclusions from the obtained result is comparable to explicit tools' support in analyzing and comparing such results. Users often require comparison across multiple runs to select appropriate models. Consequently, interpretability [17, 31] of experiment results and explainability [67] of models are essential concerns that can benefit from addressing challenges in comparing, analyzing, and interpreting outcomes of machine learning experiments.

2.2 Machine Learning Assets and Asset Management

Conventionally, the term *asset* is used for an item that has been designed for use in multiple contexts [47], such as a design, a specification, source code, a piece of documentation, or a test suite. Machine learning practitioners and data scientists often use the term *artifact* to describe different required resources during model development. These artifacts qualify as assets in the machine learning context because of the experimental nature, which requires keeping artifacts for future use. Conventional software engineering primarily deals with source code artifacts and, therefore, often has fewer asset types than the engineering of machine-learning-based systems. In contrast, machine learning includes additional artifact types, such as datasets, models, hyperparameters, and model evaluation metrics [32]. Consequently, we describe machine learning assets as individually storable units serving specific purposes in a machine learning workflow.

In the current state of practice, it is tempting to adopt traditional software engineering techniques such as VCS to address some of the highlighted asset management challenges. However, such tools were not designed to manage machine-learning-specific assets nor support the intuitive and exploratory development approach of developing machine learning components. Consequently, to address the asset management challenges, there is a need for explicit management tools and methods that offer systematic ways to 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 machine learning experiments and machine-learning-based systems in general:

Definition 1 (Asset Management)**.** The discipline of *asset management* comprises methods and tools for managing *machine-learning assets* to facilitate activities involved in the development, deployment, and operation of machine-learning-based systems. It offers *structures* for storing and tracking machine learning assets of different types, as well as *operations* that engineers 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 areas, including dataset management, model management, hyper-parameter management, process execution management, and report management.

Several classes of supporting tools are currently available for use during machine learning model development. Silva et al. [21] classify the group of supporting tools used by machine learning users into five non-exclusive categories based on their main functionality: (a) data management systems, (b) model development systems, (c) systems for the management of machine learning model lifecycle, (d) systems for the management of machine learning models, and (e) model serving systems. We briefly explain these categories and discuss their asset management capabilities in the following paragraphs.

Data management. The quality of datasets used in a machine learning model development plays a crucial role in the model's performance. Therefore, data understanding, preparation, and validation are crucial aspects of machine learning engineering. In this management area, tools (e.g., OrhpeusDB) focus on the machine learning lifecycle's data-oriented works and provide operations such as tracking, versioning, and provenance on dataset assets.

Model development. Management tools in this area focus on model-oriented works of the machine learning lifecycle. They provide supervised and unsupervised learning methods, such as classification, regression, and clustering algorithms, to generate and evaluate machine learning models. The machine learning community has mainly focused on model-oriented work, as witnessed by an extensive collection of available systems, frameworks, and libraries for model development (e.g., PyTorch, Scikit-Learn, or TensorFlow).

Lifecycle management. Management tools in this category focus on all the machine learning lifecycle stages and provide management support for all asset types produced during those stages. These include experiment management tools (e.g., MLFlow, Neptune) and pipeline management tools (e.g., KubeFlow).

Model management. These tools provide more specific support for managing already produced machine learning models. Such support includes the efficient storage and retrieval of models, model selection, and model comparison.

Model serving. Tools under this area focus on model operation. They provide efficient storage and retrieval of models to support the deployment, monitoring, and serving process. They provide information on the lineage of related assets and various evaluation performances of models (e.g., ModelDB).

There are overlapping asset management functionalities across the different categories described above. According to Silva et al.'s classification, experiment management tools—the focus of this study—fall under the category of machine-learning lifecycle management tools. Our description of the machine learning lifecycle, machine learning experiments, and assets management also apply to deep learning, a family of machine learning based on artificial neural networks [8]. In fact, some experiment management tools specialize in support for deep learning. Machine learning experiments are a core aspect of machine-learning development. Consequently, the rest of this paper focuses on the experiment management tools as our subject tools because they support users with asset management support during experimentation.

3 Methodology

We now describe our methodology for identifying and analyzing our study subjects: state-of-research and state-of-practice tools with asset management support. We describe our research questions, the data sources we used to identify our subjects, the selection criteria, and how we extracted data from the literature and tools' documentation to analyze our subjects.

3.1 Research Questions

In this study, we addressed the following research questions:

***RQ*1** *What are the machine learning assets tracked and managed by state-of-research and state-of-practice tools?*
The essential machine learning asset types used during model development include the dataset used in training machine-learning models, the scripts used in pre-processing data, and the scripts used in training and evaluating the models. However, additional information can be considered, as tools may offer various distinct management capabilities. Here, we analyzed our subject tools and tried to understand the various asset types they support.

***RQ*2** *What are the mechanisms offered for collecting the assets?*
Without dedicated tool support, users need to adopt manual and ad hoc methods to collect and store assets. Machine

learning tools offering asset management capabilities should help to reduce this manual effort. We analyzed our subject tools to identify how users can interact with them to collect the supported assets.

***RQ*3** *How are the assets stored and version-controlled?*
Similar to *RQ*2, we aim to identify the storage types offered by the tools. Here, we investigated how machine learning assets are stored relative to the tools and determined the versioning capabilities offered by the tools.

***RQ*4** *What are the management operations offered to users by the tools?*
We analyzed our subject tools to identify the asset operations that they offer to users. We expected that the tools offer management operations based on the asset management area they address or the supported asset types. For example, tools that track datasets will likely have data-related operations, while those that support models may provide operations such as evaluation and reproducibility of models.

***RQ*5** *What are the commonalities and variations between the state-of-research tools and the state-of-practice tools?*
Expanding on the answers for RQ1–RQ4, we aimed to understand the variabilities and commonalities between our subject tools as found in the research literature and in practice.

The research questions RQ1–RQ5 were triangulated from two sources—tools used in practice and tools found in research publications.

3.2 Collecting State-of-Research Tools

We conducted a Systematic Literature Review (SLR, [49]) to select the relevant literature and qualitatively analyze it to answer our research questions.

3.2.1 Selection Criteria As proposed by Kitchenham and Charters [49], we describe the inclusion and exclusion criteria used in this survey to filter and define the scope of tools that we analyzed. Our selection criteria ensured that we considered relevant literature that proposed tools with machine-learning asset management support in line with our research questions. We defined the following inclusion criteria (IC):

IC_1 Written in English.

IC_2 The paper presents a tool or prototype tool for managing machine learning experiments.

In addition, we defined the following exclusion criteria (EC):

EC_1 The paper was published more than five years ago.

EC_2 The paper does not mention machine learning or deep learning (or their corresponding abbreviations) in its title or abstract.

EC_3 The paper proposes specialized management tools with a complete focus on one machine learning asset type, e.g., dataset-specific or model-specific management tools.

EC_4 Papers about tools popularly used in practice, e.g., TensorFlow Extended, MLFlow

EC_5 Conceptualized tools, e.g., Gypscie [21]

EC_6 Literature with too few details to provide adequate answers to our research questions, e.g., [74]

The reason for exclusion criteria EC_1 was that, for this part of our methodology, we were only interested in capturing the current landscape of state-of-research tools.

3.2.2 Search Strategy For our literature search, to mitigate the chances of missing relevant literature, we relied on multiple data sources. We started with literature known to us from our expertise in the field, followed by snowballing, and lastly, searches in bibliography databases.

Knowledge ($Source_1$) As our first source, we selected publications that we knew and deemed relevant for machine learning asset management. Our knowledge and experience of machine learning and its application [1, 5, 40, 42, 43, 66, 76] guided this selection. We applied our selection criteria (described shortly) to arrive at five publications from this source. These publications are marked with a **K** in Table 1.

Snowballing ($Source_2$) Using the five selected publications from $Source_1$, we performed a backward and forward snowballing search to identify further publications concerned with our research objectives. We limited the backward snowballing search scope by publication year described in our selection criteria, which implies that the earliest publication dates of our snowballing search results were from 2016. We identified 52 related papers through this source and arrived at additional seven publications after applying our selection criteria. These publications are marked with **SB** in Table 1.

Literature Search ($Source_3$) We carried out a manual literature search from different sources. We used DBLP, ACM Digital Library, and Google Scholar as search engines, together with various search terms based on our research questions. We put in a comprehensive effort to experiment with different search terms, aiming to mitigate reliability issues and possible challenges in replicating our study. We derived these search terms from known relevant literature (as per $Source_1$ and $Source_2$). Beyond the final set of terms (discussed below), we experimented with additional ones iteratively. First, we experimented with the terms "asset" and "asset management." However, these did not produce any relevant results: in the context of machine learning, "asset management" is a new term coined in our work. Second, we experimented with the keywords "process," "operation," and "platform." These searches only lead to redundant results already covered by the known literature, as well as unrelated tools that do not support experiment management and were, therefore, excluded.

Using the combination of results from $Source_1$ and $Source_2$, we identified final key terms such as "lifecycle management," "experiment management," and "model management." We based our literature search queries on these terms. First, we carried out a title search on DBLP using the query

> *Q1 - ("machine learning" | "deep learning") & ("lifecycle" | "model" | "experiment" | "data" | "metadata") & "management."*

From this search, we obtained 75 matches. In contrast to DBLP, the ACM Digital Library allows searching on abstracts as well. We queried it using:

> *Q2 - ("machine learning" OR "deep learning") AND ("lifecycle" OR "model" OR "experiment" OR "data" OR "metadata") AND "management."*

From this search, we obtained 586 results. A title scan was carried out on search results to select literature relevant to our research objectives. This was followed by an abstract and content scan when necessary to determine the relevancy of the literature. We found no new publication after applying our selection criteria.

3.3 Collecting State-of-practice Tools

We were also interested in analyzing tools used in practice for their asset management support. Most tools do not have related scientific publications; consequently, we collected the relevant tools from the grey literature.

Table 1. The state of research subject tools

Subjects	Source	Ref	Venue
ModelHub	K	[54, 55]	ICDE
Runway	K	[79]	MLSys
ModelDB	K	[82]	MoD
Deep-water	K	[27]	SoftwareX
ModelKB	K	[28, 29]	MoD
DeepDiva	SB	[2]	ICFHR
Declarative	SB	[70, 71]	NIPS
Vamsa	SB	[61]	KDD
DLHuB	SB	[18]	IPDPS
ModelOps	SB	[39]	IC2E
Pdmdims	SB	[65]	CLOUD
CANDLE	SB	[86]	CAFC

Table 2. The state of practice subject tools

Cloud Service	Software
Neptune.ml (netptune.ml)	Datmo (github/datmo)
Valohai (valohai.com)	Feature Forge (github/machinalis)
Weights & Biases (wandb.com)	Guild (guild.ai)
Determine.ai (determined.ai)	MLFlow (mlflow.org)
Comet.ml (comet.ml)	Sacred (github/IDSIA)
Deepkit (github/deepkit)	StudioML (github/open-research)
Dot Science (dotscience.com)	Sumatra (neuralensemble.org)
PolyAxon (polyaxon.com)	DVC (dvc.org)
Allegro Trains (github/allegroai)	Codalab (worksheets.codalab.org/)

Access date: Aug. 2021

3.4 Selection Criteria

Our selection criteria here ensured that we consider the most relevant tools used in practice for machine learning asset management, in line with our research questions. We defined the following inclusion criteria (IC):

IC_1 Well-documented tools with enough available details to address our research questions

IC_2 Tools used for management of different asset types involved in machine learning

IC_3 Tools with meaningful prominence measured by at least 25 GitHub stars

In addition, we defined the following exclusion criteria (EC):

EC_1 Specialized management tools such as dataset-specific or model-specific management tools.

EC_2 General or multipurpose computational workflow or provenance management tools that are not specifically designed for machine learning

EC_3 General machine learning framework or model development tools

3.4.1 Search Strategy For selecting tools with machine-learning asset management support used in practice, we used the Google search engine as our primary data source. Our query was:

> *("machine learning") AND ("artifacts" OR "experiments" OR "lifecycle") AND ("provenance" OR "versioning" OR "tracking") AND "management" AND ("framework" OR "tool" OR "platform").*

From this search, we obtained 181 results. It is important to note that, as a well-known phenomenon of Google search, we obtained an initial result of over two million counts, which later decreased to 181 after navigating to the last result page. After applying our selection criteria, we arrived at 17 different tools. To ensure that we have selected the most relevant state of practice tools, we identified surveys [27, 46, 83] of related tools and checked their surveyed tools against our selection.

We included one additional tool—Codalab—based on personal experience, resulting in the final 18 tools used in practice.

The final result of the overall selection process is presented in Tables 1 and 2.

3.5 Data Analysis

This study aimed to characterize our subject tools using features [9] and to represent them in a feature model [48, 62]. For the selected state-of-practice tools, we considered relevant information found in their publicly available documentation. In a few cases, we had to test the tools for their available functionality when needed practically. For each state-of-research tool, we considered the information provided in the corresponding literature to answer our research questions.

The analysis process was divided into different stages to study all our subjects' features and build a feature model. First, we performed an initial analysis of a single state-of-practice tool to identify its supported machine learning asset types, the asset collection approaches, the storage options, and supported asset operations. We partly established the terminologies to be used in our models. We also arranged the established features according to perspectives based on the subject tools' support and use-cases. This led to a baseline version of our feature model. Second, we iteratively evaluated additional subjects while modifying terminologies and the model structure to accommodate variations from the new tools being assessed. After completing the analysis for state-of-practice subjects, we carried out a similar analysis for the state-of-research subjects.

One author read the literature selection and mapped their features to those found in the latest state-of-practice feature model, leading to extensions of the feature model for those features that could not be mapped that way. Finally, at the end of the final iteration, all authors met to review the latest structure for direct feedback on terminologies used and the outcome feature model.

4 Asset Management Features

We propose a feature model——outlined in Figures 3 to 7——to characterize and describe the machine learning asset types and the management support found in our subjects. In what follows, we refer to the features using a `typewriter` font style. The top-level features—`Asset Type`, `Collection`, `Storage`, and `Operation`—capture the core functionalities of the subjects in our study and correspond to our research questions RQ1–RQ4, while Table 3, Table 4, and the associated text answer the research question RQ5. As described in the following subsections, `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.

4.1 Asset Type (RQ1)

Unlike traditional software engineering, whose primary asset type is source code, machine learning has more diversified asset types, such as datasets, hyper-parameters used in training, transformation codes, trained models, and evaluation metrics. The supported types vary between our subject tools, with different levels of support. For example, some subjects only track explicitly declared asset metadata, while others can automatically store assets. On a high level, the primary asset types commonly associated with machine learning are datasets, source code, and generated models.

However, our subject tools provide more specialized support for different asset types to users. Hence, 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. For implicit support, a tool could allow the user to specify asset types by declaring meta-data, such as available column labels, dimensions, and data sources. Both explicit and implicit supports allow the subject tools to offer operations such as tracking, which can help answer provenance questions, such as "what is the source of the dataset?"

On the other hand, some subject tools store the assets and their attributes internally and offer operations such as asset versioning. As shown in Fig. 4, our analysis identifies features `Resources`, `Software`, `Metadata`, and `ExecutionData`

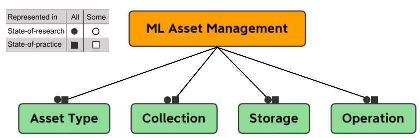

Fig. 3. Main machine learning asset management features.

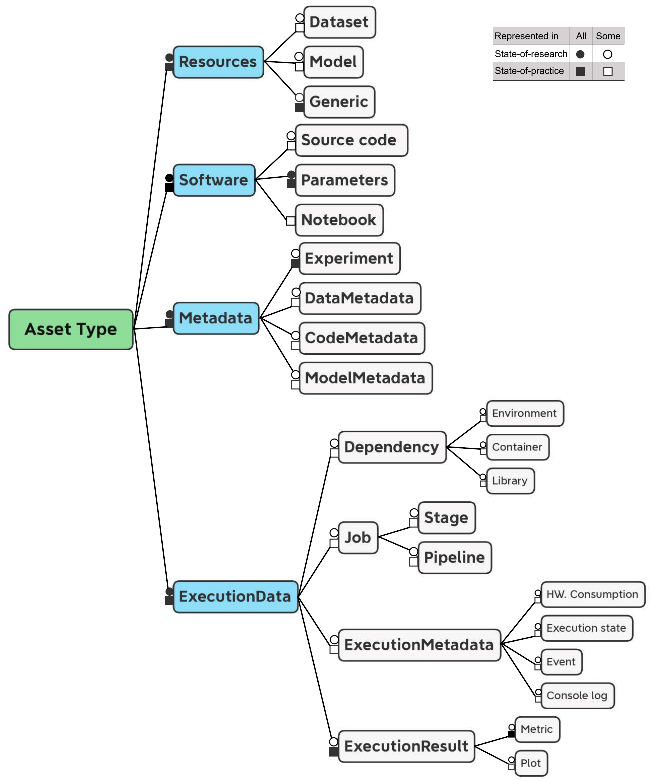

Fig. 4. Asset Types: A representation of the data types tracked by the subjects under study.

as the sub-features of `Asset Type`. `Resources` represent the core asset types of the machine learning workflow. In contrast, `Software` refers to the implementation responsible for changing the states of these assets. The `metadata` and structured information about `Resource` and `Software` are detailed as well. The `Executiondata` are information about the execution of machine learning runs and the outcome of such runs.

4.1.1 Resources: `Resources` are commonly referred to as *artifacts* by many of the subject tools. We describe `Resources` as the asset types required as input or produced as output from a stage of the machine learning workflow (see Fig. 1). We identified features `Dataset`, `Model`, and `Generic` as the sub-feature of `Asset Type`. We identified `Dataset` and `Model` as the most critical resource types. The data-oriented stages of the machine learning workflow usually require a dataset as input and output a transformed version of it. The model-oriented stages deal with models as output and input data for the training and testing stages. The DevOps stages also involve serving models and ingesting data for learning inferences. We classify other assets under `Resource` as `Generic`. The tools often track such generic resources as arbitrary resources. For example, some subject tools such as MLFlow and Neptune provide a *log_artifact* method to track any arbitrary file used during an experiment. Some subject tools track resources through their metadata and do not necessarily support the actual resource type (e.g., pointer information to location and the hash of data stored locally or on remote cloud storage systems). For example, DeepDiva [2] manages datasets by providing paths to the file directory containing the datasets. Other subject tools provide at least one dedicated method to track, provide resource-type-specific support, and sometimes internally store the resources. We only consider the latter to support resources fully. As an example, the subject PolyAxon allows users to use *log_dataframe* to track and store a dataset asset type.

Dataset: The feature `Dataset` is available for subjects that identify datasets as an asset type. Data is the core asset type in machine learning since its quality plays a major role in the performance of a machine learning component. Hence, the machine learning workflow stages of data collection, data transformation, feature extraction, model training, and evaluation are data-dependent. The presence of the feature `Dataset` implies that the subject tool supports the tracking of datasets along with experiment-associated assets to provide dataset provenance.

We found that most of our subjects, such as ModelHub, Runway, Deep-water, Neptune, and Sacred, require users to explicitly track operations, such as pre-processing or feature engineering carried out on the datasets. The supported dataset usually ranges from database-based data to file-based data such as spreadsheets, CSVs, or streaming datasets. For example, Deep-Water [27] allows users to provide CSV-based datasets as the training or test datasets, while tools such as Vamsa [61] support Panda[1] Dataframes and automatically collect information about the Dataframes from Python scripts.

Model: Machine learning models are created by learning from datasets using learning algorithms. The latter are typically provided by machine learning development frameworks, such as TensorFlow[2] and SciKit-Learn[3]. The feature `Model` is available for the subjects that identify models as an asset type. Like datasets, most subjects support direct or indirect tracking or storing of models and their metadata to support asset management operations such as provenance analysis. Support for the feature `Model` is required for certain operations, such as the model comparison from different experimental runs or management of model evolution through different stages. Some subjects, such as ModelHub [54, 55], offer model storage and its efficient retrieval as their primary functionality, while other subjects, such as Runway [79] and ModelKB [28, 29], primarily track models and their metadata for post-experiment result analysis.

Generic: The presence of the feature `Generic` indicates support for tracking and managing any resource files used during machine learning workflows, typically for resources required along with datasets and models. Examples include credentials for authentication in external services that host other resources. In addition, several subjects lack dedicated support for tracking `Dataset` or `Model` types; instead, they provide a "one-size-fits-all" tracking of resources. Consequently, subjects with feature `Generic` can track all binary files of any type that are required or generated during an experiment, without differentiating them or providing dedicated operation beyond storage.

4.1.2 Software: Traditional version control systems (VCSs), such as Git, are essential source code management tools. The engineering of machine-learning-based systems partly involves managing the source code used to implement the machine learning operations. Users often try to balance managing assets with traditional VCSs versus using alternative approaches tailored toward machine learning workflow. The exploratory working style of data scientists and machine learning practitioners challenges the effective use of traditional VCSs to manage assets when engineering machine learning systems [6]. The pain points of data scientists and machine learning practitioners when versioning their source code also motivate the need for a different approach to machine-learning software asset management. The feature `Software` refers to implementation assets of the machine learning process, such as `SourceCode`, `Parameter`, and `Notebook`, which are typically involved in the implementation of stages of the machine-learning workflow. This feature heavily relies on the

[1] https://pandas.pydata.org
[2] http://tensorflow.org
[3] http://scikit-learn.org

machine learning model development tools (e.g., SciKit-Learn,[4] PyTorch,[5] TensorFlow,[6] and Keras[7]) that provide a collection of general machine learning techniques to users.

Source Code: This feature represents support for text-based files with implementation to carry out specific machine learning operations. Managing source code (or scripts) is generally less challenging than notebook formats for functional and large-scale engineering of machine-learning-based systems because of available IDEs to support code assistance, dependency management, and debugging [19]. Source code consists of text-based files; therefore, it is easily version-controlled using traditional VCSs. Consequently, about 70% of our subjects track source code via metadata, which we represent by `CodeMetadata`. Other tools provide an integrated source code management approach: for example, DVC builds on Git and provides new commands tailored toward managing source code along with other machine learning assets.

Parameter: Hyper-parameters are parameters utilized to control the learning process of a machine learning algorithm during the model training (e.g., learning rate, regularization, and tree depth). Some subjects track hyper-parameters to facilitate the analysis of experiment results. Some subjects (e.g., Comet, Polyaxon, and Valoh.ai) provide hyper-parameter tuning and search features to facilitate the model-oriented stages of a machine learning workflow. In addition to hyper-parameters, the asset type `Parameter` also represents other configurable parameters that users may require to influence their machine learning workflow.

Notebook: Similar to source code, notebooks contain the implementation to carry out specific machine learning tasks. `Notebooks`, written in multiple execution cells, are usually used for small-scale, exploratory, and experimental machine learning tasks, where it is difficult to achieve acceptable software engineering practices, such as modular design or code reuse. Notebooks (e.g., Jupyter [50]) are crucial for reproducible machine learning workflows that require literate and interactive programming. The feature `Notebook` indicates the support to track notebooks as an asset type. The feature `Notebook` is available in 6 state-of-practice tools, and none of the state-of-research tools provides explicit management options for notebook formats.

4.1.3 Metadata: Conventionally, metadata allows the semantic description of entities. In our context, as a sub-feature of the `Asset Type`, `Metadata` represents the descriptive and structural static information about core assets, including machine learning experiments and their associated resource assets. These include information such as name, version and URI. We identified the sub-features of `Metadata` as `Experiment`, `DatasetMetadata`, `CodeMetadata`, and `ModelMetadata`.

Experiment: The feature `Experiment` represents the main asset type with which other asset types are associated. It is the core abstraction of experiment management tools. Other assets that our subjects track are often associated with an `experiment`. This feature presents the book-keeping record of different runs performed for a machine learning experiment.

DatasetMetadata: This feature represents support for tracking the state of datasets used in a machine learning workflow. It represents dataset-related information, such as data location, data origin, version id, data schema, and data frame structure.

[4]https://scikit-learn.org/stable/
[5]https://pytorch.org
[6]https://www.tensorflow.org
[7]https://keras.io

CodeMetadata: Since source code is traditionally well handled by existing VCSs such as Git, many of our subject tools allow users to manage `SourceCode` through the traditional VCSs by tracking their metadata, such as the repository name, link, and commit hash.

ModelMetadata: This feature represents support for tracking the metadata of models generated or used in a machine learning workflow. It represents model-related information, such as model author, creation date, input attribute schema, and details about the model-generating source code.

4.1.4 ExecutionData: This feature represents execution-related data that the subject tools track explicitly or automatically before or during the execution of a machine learning experiment. We identified `dependency`, `job`, `ExecutionMetadata` and `ExecutionResult` as sub-features of `ExecutionData`.

Dependency: Tatman et al. [78] reveal that sharing an environment with source code and dataset provides the highest level of reproducibility. Consequently, some subjects, which support experiment reproducibility, track and manage required dependencies to reproduce models or rerun experiments. The feature `Dependency` helps users track data on systems' `Environment` information, such as environment variables, host OS information, or hardware details; `Container` such as Docker[8] containers; and required `Libraries` and versions used for an experiment. Some subjects leverage unique environments and dependency management systems such as Conda[9] to track and manage dependency information.

Jobs: The feature `Job` represents the execution instructions of a machine learning experiment and how associated assets defined by `Resources` and `Software` should be used during execution. There is usually a 'one-to-one' or 'one-to-many' relationship between an `Experiment` and its `Jobs`. We describe the feature `Job` as a `Stage` or a `Pipeline`, where a `Stage` represents a single stage of the workflow, and `Pipeline` represents a sequence of multiple stages. Listing 1 shows the representation of a stage and pipeline in the subject DVC. The use of Command Line Interface (CLI) commands as execution instruction by some subjects also qualifies as a form of `Job` representation.

- A `Stage` is a basic reusable phase of a machine learning workflow, as illustrated in Fig. 1. Stages are defined with pointers to their required assets, such as the source code, parameters, and input resources, such as datasets.
- A `Pipeline` represents a reusable relationship between multiple stages to produce machine learning workflow variants, as described in Fig. 1. Subjects with support for workflow allow users to define pipelines as dependency graphs, which use input and output resources as dependencies between stages. In Listing 1, a dependency graph is illustrated with a *featurize* stage, which depends on the output of the *prepare* stage.

ExecutionMetadata: This feature represents all information about the execution process that the subject tools capture while the experiment execution is ongoing. Data commonly tracked as `ExecutionMetadata` include hardware consumption, execution states, events, and console logs. Examples of the information under hardware or system consumption include CPU, GPU, and memory utilization for experiment tasks, while execution states indicate the progress or status of ongoing or completed experiments. Events may be used for notifying users of essential activities during model training, especially for long training processes. Console logs such as *stderr* and *stdout* are captured when executing machine learning experiments.

ExecutionResult: This feature represents the assets generated as the results of an experiment or different experiment runs. `ExecutionResults` are often associated with the model training stages of an experiment. This refers to the

[8] https://www.docker.com
[9] https://docs.conda.io/en/latest/

```yaml
stages: # Pipeline
  prepare: # Stage
    cmd: python src/prepare.py data/data.xml
    deps:
    - data/data.xml
    - src/prepare.py
    params: # Configuration
    - prepare.seed
    outs:
    - data/prepared
  featurize: # Stage
    cmd: python src/featurization.py data/prepared data/features
    deps:
    - data/prepared
    - src/featurization.py
    params: # Configuration
    - featurize.max_features
    - featurize.ngrams
    outs:
    - data/features
```

Listing 1. An example representation of a `Pipeline` with two `Stages` in our subject DVC. This example defines two stages of an experiment as *prepare* and *featurize*, and describes the entry point, dependency, parameters and outputs of each stage of the experiment pipeline.

evaluation `Metrics` and `Plots` that are tracked in different forms based on machine learning tasks (e.g., sensitivity or ROC values for classification tasks; MSE, MAPE, or R^2 for regression tasks). Subjects provide specific methods such as *log_metrics* or *log_artifact* (see Listing 2) to track performance metrics, including accuracy and training loss. These metric values are either tracked as single value metrics or a series of metrics values in a training loop. Subjects employ different approaches for collecting series values. For example, invoking *log_metrics("metric-name", "value")* multiple times in a training loop will collect the *"metrics-name"* results as a series of data which can be summarized in dashboards. Subject tools supporting metrics series value include Neptune, Valohai, Wandb, Determine AI, and DotScience. For model training and data-oriented stages, assets of type `Model` and `Dataset` are the result indicating a relationship between the feature `ExecutionResult` and feature `Resources`.

What asset types are tracked by the subject tools? (RQ1)

Our subject tools support software assets (including source code, notebooks, and parameters), resources (including datasets, models, and generic resources), and various metadata. The commonly supported asset types are generic files, parameters, experiment metadata, and execution results. Many of the subject tools support free-form generic asset types and metadata, which implies limited out-of-the-box support for machine-learning-specific concerns.

4.2 Collection (RQ2)

The feature `Collection`, shown in Fig. 5, represents the options provided by our subjects to track the asset types we identify in Section 4.1. The feature `Intrusiveness` shows the level of the explicit declaration required to collect the assets. The feature `Location` represents the location where the subject tools collect the assets.

```
# Collect a CSV-based dataframe object, 'df' using PolyAxon
log_dataframe(self, df, name, content_type='csv', step=None)

# Collect a model weights using Neptune
my_model = ...
touch.save(my_model, 'my_model.pt')
neptune.log_artifact('my_model.pt', 'model_checkpoints/my_model.pt')

# Collect a single metric value using Neptune
neptune.log_metric('test_accuracy', 0.76)

# Collect accuracy per epoch as series value using Neptune
for epoch in range(epoch_nr):
    epoch_accuracy = ...
    neptune.log_metric('epoch_accuracy', epoch_accuracy)

# Collect local variable of decorated function as experiment configuration using Sacred
@ex.config
def exp_parameters():
    ccp_alpha = 0.1
    n_estimators = 50
```

Listing 2. Examples of asset collection from source code using subject tools PolyAxon, Neptune, and Sacred. PolyAxon and Neptune provide dedicated functions to track specific asset types like datasets, models, and metrics, while Sacred tracks variables of a decorated function as experiment parameters.

Intrusiveness: This sub-feature describes the amount of instrumentation required by users to track assets. The `Intrusive` collection is invasive and requires users to add special instructions and API calls in source code or define special configuration files to track desired assets. In contrast, the `Non-Intrusive` collection automatically tracks or logs assets without the need for explicit instructions, API calls, or special files. Subjects with support for non-intrusive asset collection, such as Neptune, MLFlow, Deep-Water [27] and ModelKB [28, 29], usually support a limited number of machine-learning frameworks, such as Sci-Kit Learn, PyTorch, and TensorFlow. These subjects are designed to recognize and interface with the supported frameworks to directly collected specific assets, such as hyperparameters, models, and evaluation metrics.

Location: This sub-feature describes where the tool is instructed to collect assets. It can be collected from `SourceCode` using predefined APIs, passed as arguments from `CLI`, extracted from execution `Logs` by parsing it, or read from instrumented `File system`. The common collection point across the considered subjects is `SourceCode`, where the subjects provide a library and API that users invoke to log desired assets within source code implementation. For example, Listing 2 shows how the subject PolyAxon and Neptune are instructed through source code to collect datasets, models, and performance metric assets. For collection at the `CLI`, subjects allow users to specify pointers to assets as command arguments. For example, references to asset files can be passed via CLI. For the collection approach using `Logs`, subjects collect assets by parsing the log output of experiment executions. For example, `ExecutionResults` such as model performance metrics can be automatically extracted from logs generated by model training runs. With the `FileDirectory` approach, subjects monitor assets from structured or instrumented file systems. Assets collected through this method are usually `Non-Intrusive` since the tools automatically track changes to assets in the target file directories.

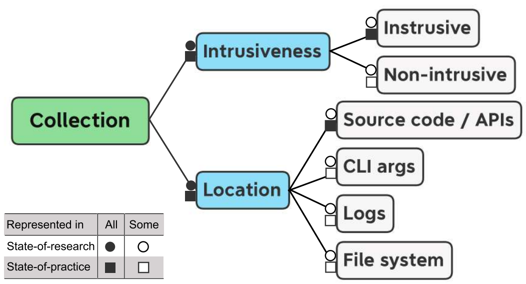

Fig. 5. Collection feature model: A representation of collection features used in tracking the asset types described in Section 4.1.

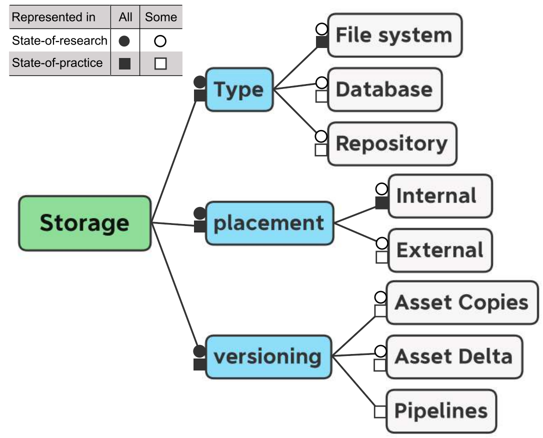

Fig. 6. Storage feature model: A representation of the storage feature identified in the subjects under study.

Some subjects allow users to specify details of asset location in special configuration files. For example, in Listing 1, the DVC configuration file specifies the location of assets, such as datasets, source code, dependencies, and parameters.

How are assets collected? (RQ2)

The subject tools support both intrusive and non-intrusive asset collection methods. Assets are collected intrusively through source code and otherwise, through CLI arguments and configuration files to collect assets from logs and instrumented file systems. Subject tools offer collection methods specific to supported asset types or generic methods for collecting assets of any type.

4.3 Storage (RQ3)

The feature `Storage` describes how the assets are stored and the versioning type supported by the subject tools. Figure 6 shows the sub-features of `Storage`.

Storage Type: The feature `Storage` is fundamental for our subjects, especially for cloud-based services, which provide cloud storage capabilities for machine learning assets. 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. The `Database` type provides a more structured internal storage option using existing database systems such as RDBM. The `Repository` type represents storage with version control support. Some subjects, such as ModelHub [54, 55] and DVC, provide custom VCSs with similar functionality like general VCSs, such as Git. It is a common practice to store source code in repositories. Consequently, many of our subjects delegate the storage of assets, such as source code, to third-party repositories.

Limitations on storable asset sizes are usually not explicitly addressed in our data sources. The publications on state-of-research tools often focus on their proposed functionality. The lack of explicit attention to storage size may be attributed to data storage being cheap [23]. Similarly, many state-of-practice tools offer paid versions where users can pay to extend storage sizes according to their needs. For local storage, it is the user's responsibility to manage the availability of sufficient storage.

Storage Placement: The placement of stored assets can either be `Internal` or `External` in relation to the subject tools. For assets stored internally, the subjects store and fully manage the assets. `External` indicates that assets are not stored within the subject tool and are usually stored remotely. Externally stored assets are usually tracked through identifier pointers and can be transferred or fetched for processing on demand. This option is suitable for large files and scenarios where users require easy access from cloud-hosted services, such as notebooks and cloud computing infrastructure. In many cases, subject tools offer ways to store assets internally and externally. For example, `Metadata` are commonly stored internally, while `Resources` are stored externally.

Versioning: The support for `Versioning` is required to track the evolution of assets by keeping versioned assets during the machine learning workflow. Traditional VCSs work well for typical source code repositories due to code files being small and text-based, but are not ideal for large datasets and models. `Versioning` represents the versioning strategies used in our subject tools. This feature is supported either by storing copies of assets for each time they are modified (`Asset copies`), storing the deltas between assets for space efficiency (`Asset delta`), or versioning pipeline metadata to reproduce assets (`Pipelines`). Storing copies of assets is an easy-to-implement versioning strategy, which is employed for direct copies of assets with different naming conventions, such as semantic versioning [52]. However, storing copies of assets is highly inefficient due to the ratio of changes to additional storage demand, especially in large files and assets. Instead, storing the delta between different asset versions offers an efficient alternative. Rather than storing and versioning `Resource` assets used, the `Pipeline` versioning only stores and versions pipeline metadata, which can later recreate derived assets on demand by replicating a particular experiment run.

How are assets stored? (RQ3)

The assets are either stored in file systems, databases, or repositories, either internally or externally, while assets are version controlled by storing copies of assets, the delta between changes, or versioning pipelines to recreate derived assets. Asset storage support ranges from small cases requiring storage on local systems to large-scale projects requiring remote storage infrastructure for large-sized assets.

4.4 Operations (RQ4)

We identify several primary operations supported by our subjects and represent them by the feature `Operation`. The `Track` and `Explore` operations are common features supported by all subjects. Figure 7 shows the sub-features of `Operation`.

Track: Tracking of machine-learning assets is the core feature offered by the subject tools. Our subject tools track either assets or metadata about them. For intrusive collection, users choose what assets to track during machine learning experiments, whereas for non-intrusive collection, tools automatically track specific assets from supported machine learning frameworks. The subject tools organize the tracked assets and metadata to help users address different concerns such as traceability raised in Section 2.1.1.

Version: The feature `Version` indicates support for versioning-related operations similar to the conventional VCS like Git. For example, where `Tracking` simply captures, and stores selected assets during experiment runs, `Versioning` is a

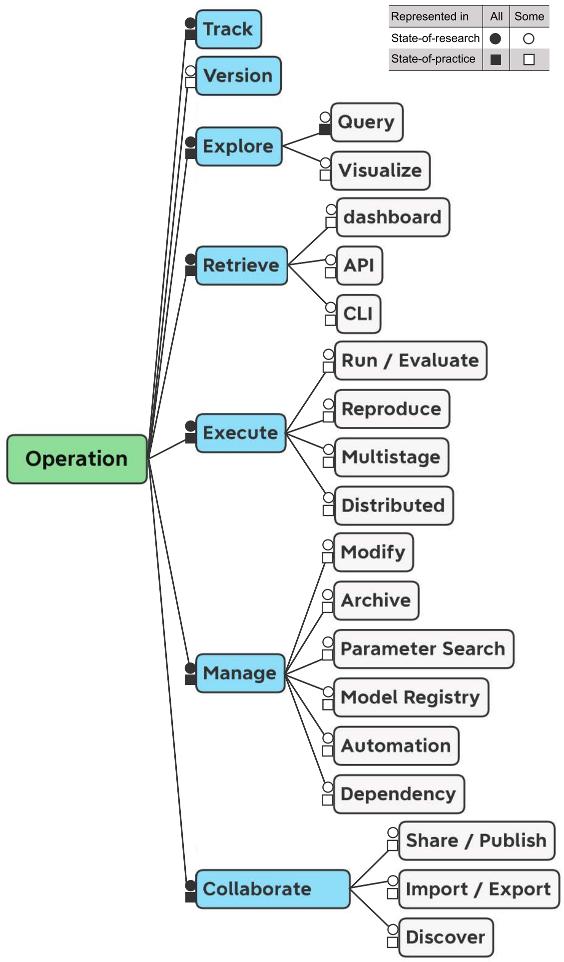

Fig. 7. Operation feature model: A representation of operations offered by the subjects under study.

step beyond tracking assets, as it provides Git-like operations such as *commit*, *revert*, and *branch* to create and recover checkpoints of assets. Models, datasets, and pipeline metadata are the most commonly supported assets for versioning. Essential versioning support allows users to commit a new version of assets and revert to earlier versions.

Explore: The feature `Explore` represents operations that help derive insight or analyze assets collected from completed machine learning 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 pipelines, parameters, and performance metrics at different points in time.

Retrieve: Retrieving stored assets for further use is an essential operation. Whereas most of our subject tools allow users to retrieve stored assets for post-model-creation analysis, such as traceability, retrieving assets is important to

Table 3. Asset Types, Collection and Storage Comparison

			State-of-Practice																		State-of-Research											
		Subjects	DVC	Codalab	Sumatra	StudioML	Sacred	MLFlow	Guild	Feature Forge	Datmo	Allegro Trains	PolyAxon	Dot Science	Deepkit	Comet.ml	Determine.ai	Weights & Biases	Valohai	Neptune.ml	CANDLE	Pdmdims	ModelOps	DLHuB	Vasma	Declarative	DeepDiva	ModelKB	Deep-water	ModelDB	Runway	ModelHub
Asset-Type	Resources	Dataset	●							●			●	●	●	●	●	●	●	●		●	●	●	●				●	●		
		Model	●					●	●			●	●	●	●	●	●				●	●	●	●	●			●		●	●	●
		Generic	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				●							●	●
	Software	Source code	●	●	●	●	●	●	●		●	●		●		●		●	●	●	●	●	●	●	●				●			●
		Parameter	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
		Notebook		●					●		●	●		●		●		●		●												
	Metadata	Experiment	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●						●		●	●	●	●	
		Datametadata		●				●		●	●	●	●	●		●	●	●	●				●	●	●	●	●	●			●	
		Codemetadata	●	●	●		●	●	●			●	●	●	●	●	●	●	●	●						●	●	●			●	●
		Modelmetadata						●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●		●	●	●
		Dependencies		●		●	●	●	●		●	●	●	●	●	●	●	●	●				●	●		●	●	●	●			
		Stage	●	●				●	●		●	●	●	●	●		●		●				●	●		●						
		Pipeline	●			●		●	●			●	●	●	●		●		●		●		●	●		●			●	●		
	Exec. Data	Exec. metadata		●	●	●	●	●	●	●		●	●	●	●	●	●	●	●	●			●				●		●		●	●
		Exec. result	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●		●		●	●	●	●	●
Collection	Intrusiveness	Intrusive	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●			●		●	●	●	●	●
		Non-intrusive		●	●	●		●				●	●	●		●		●		●					●	●		●		●		
	Location	Src. code / api	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●			●			●		●	●	●	●	
		CLI args	●	●	●	●		●	●		●	●	●		●	●	●		●													●
		Logs					●					●								●												●
		File system	●		●			●			●	●	●	●			●		●				●		●							
Storage	Type	File system	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●								●			●	
		Database			●		●	●		●											●	●	●	●		●		●	●	●		●
		Repository	●	●										●																	●	●
	Placement	Internal	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●		●				●	●	●
		External	●			●		●	●			●	●	●		●	●	●	●	●						●					●	
	Versioning	Asset copies		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●				●			●			●
		Asset delta	●											●																		●
		Pipelines	●									●	●	●	●		●		●													

support the reuse of machine learning assets. For example, derived assets from an earlier experiment run (e.g., transformed datasets) can be fetched for further transformation or used for evaluating a model from another experiment run. The most supported approach to `Retrieve` or access stored assets is by GUI-based `Dashboards` to explore, visualize, and compare experiment results. Other means of access include `API`, which provides REST interfaces or programming language APIs to access assets stored by the subject; feature `CLI` exists for subjects that provide CLI commands for asset management.

Execute: The feature `Execute` indicates operation support that allows subject tools to manage the execution of machine learning experiments. The tools allow users to specify the entry-point to invoke their experiments. The features `Run` and `Reproduce` allow the execution of new and the reproduction of prior experiments, respectively. For example, some subjects, such as ModelKB [28, 29], offer the feature `Reproduce` by providing functionality to package machine learning models along with their metadata and required resources into a reusable format. The feature `Multistage` indicates support for multistep execution, which is required for tools supporting the execution of multistage machine learning experiments. Similarly, the feature `Distributed` indicates support of the tool to leverage parallel and distributed computing resources when executing machine learning experiments. `Multistep` and `Distributed` features are usually found in tools with support for managing and automating the execution of experiment pipelines.

Manage: While some subjects treat specific asset types as immutable, where any update to existing assets results in new versions, other subjects supporting the feature `Modify` allow some level of modification or removal to revise already stored assets. The amount of data generated from machine learning experiments over time can be significant. Consequently, some subject tools allow users to archive assets, such as models and datasets. We represent this feature as `Archive`. We represent the support for hyperparameter search and tuning by the feature `Parameter Search`. Subjects support this feature by providing common parameter search techniques, such as exhaustive space search, simple/multiple gradient descent, random search, list search, and range search. Model-store-specific operations often target model management between different lifecycle stages. For example, they support users in retrieving models for testing, serving, and deploying efficiently. We represent this operation by the feature `Model Registry`. Some subjects support automating the machine learning experiments or certain aspects of the machine learning lifecycle. We present this by the feature `Automation`. Similarly, the feature `Dependency` represents the presence of dependency management, where dependencies are often implemented as direct acyclic graphs.

Collaborate: This feature represents the presence of collaboration features, which are often targeted at teams that need to share assets and obtain experiment results among the team members. Users can `Share`, `Publish`, `Export`, `Import`, or `Discover` machine learning experiment outcomes or other assets.

What are the supported operations? (RQ4)

At a minimum, all subject tools allow users to track assets, while 93% allow users to explore experiment assets using queries and visualization for comparison and insights on experiment results. Other operations enable users to retrieve assets manually or programmatically, reproduce experiments, manage stored assets, and collaborate by sharing or publishing assets. Often the operations address one or multiple asset management challenges, including reproducibility, interpretability, and collaboration.

4.5 Comparison of State-of-research & State-of-practice Tools (RQ5)

We present the comparison of the features found in both state-of-practice and the state-of-research subjects in Tables 3 and 4. It is important to mention that state-of-research tools can be expected to have fewer features by design, since they usually focus on a specific research problem, such as metadata tracking [70, 71]. The state-of-practice subjects

Table 4. Operations Comparison

Group	Subjects	Track: Track	Version: Version	Explore: Query	Explore: Visualize	Retrieve: Dashboard	Retrieve: API	Retrieve: CLI	Execute: Run	Execute: Reproduce	Execute: Multistage	Execute: Distributed	Manage: Modify	Manage: Archive	Manage: Param. search	Manage: Model registry	Manage: Automation	Manage: Dependency	Collaborate: Share/Publish	Collaborate: Import/Export	Collaborate: Discover
State-of-Practice	DVC	●	●	●	●		●	●	●	●	●		●				●	●	●	●	
	Codalab	●	●	●		●	●	●	●	●		●	●					●	●	●	●
	Sumatra	●		●		●			●	●		●	●							●	
	StudioML	●		●	●			●	●						●						
	Sacred	●		●			●	●	●								●				
	MLFlow	●		●	●	●	●		●	●	●	●		●		●	●	●	●	●	
	Guild	●	●	●	●	●	●	●	●	●	●		●		●		●	●	●	●	●
	Feature Forge	●		●			●		●			●									
	Datmo	●	●	●			●	●	●	●			●								
	Allegro Trains	●		●	●	●	●	●	●	●	●	●	●	●	●		●	●	●		●
	PolyAxon	●		●	●	●	●	●	●	●	●	●			●	●	●	●	●		●
	Dot Science	●	●	●	●	●	●		●	●	●	●			●	●	●	●	●		●
	Deepkit	●		●	●	●			●	●	●	●					●	●			
	Comet.ml	●		●	●	●	●	●	●	●		●			●	●	●	●	●		●
	Determine.ai	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	Weights & Biases	●		●	●	●	●	●	●	●	●	●			●	●	●	●			
	Valohai	●		●	●	●	●		●	●	●	●			●	●	●	●			
	Neptune.ml	●		●	●	●	●						●			●			●		●
State-of-Research	CANDLE	●									●	●			●						
	Pdmdims	●		●	●															●	
	ModelOps	●	●	●					●	●	●	●					●	●			
	DLHuB	●	●	●			●	●	●	●	●	●					●		●		
	Vasma	●	●																		
	Declarative	●		●		●			●	●									●	●	
	DeepDiva	●		●	●	●			●	●					●						
	ModelKB	●		●	●	●			●	●											
	Deep-water	●	●	●		●			●	●	●	●			●						
	ModelDB	●		●	●	●								●							
	Runway	●		●	●	●							●	●							
	ModelHub	●	●	●		●	●	●	●					●	●	●			●		●

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 machine learning 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 machine learning experiments and their associated assets. Although all subject tools support static metadata assets, we observe the presence of more metadata types for the state-of-practice subjects. Roughly half of the subjects in each group support representation of workflows as stages and pipelines. Execution results are supported and tracked by 58% of the state-of-practice tools and 50% of the state-of-research tools, while fewer subjects track execution metadata.

The asset collection method supported by most subjects requires users to instrument their source code. 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 the use of instrumented file systems. The tracked assets are mostly stored in file systems for the state-of-practice subjects, while the state-of-research tools primarily employ databases. Only a few subjects, such as ModelHub, Runway, DVC, and DotScience, employ custom or reuse traditional version control for internal asset storage. The common versioning strategies used in both groups are based on creating copies of assets. The use of pipeline versioning is also prominently found among the state-of-practice tools.

All state-of-practice and state-of-research subjects allow users to track the supported asset types. A few subjects from both categories also support versioning operations similar to the conventional version control system operations. Subjects in both categories offer support to query and visualize assets, with most subjects providing access via web-based dashboards. Most subjects under the state-of-practice tools also offer access to assets via command-line interface and programming APIs. Almost all state-of-practice and about half state-of-research subjects offer execution-related operations. Similarly, operations for modifying specific stored assets are only supported by a few subjects state-of-practice, whereas support for archiving is available by a small number of subjects in each category. Parameter search and model registry operations are predominantly found in the state-of-practice subjects. Support to automate the execution of machine learning workflow and manage the asset dependencies is mostly available in state-of-practice subjects. Collaboration-related operations such as sharing, publishing, importing, exporting, and discovering assets from other users are supported by half of the state-of-practice subjects and only one subject from state-of-research.

Differences between State-of-research and State-of-practice Tools? (RQ5)

There are similarities in the assets types, collection, storage, and operations supported across subjects in the state-of-practice and the state-of-research subjects. A notable difference between the two groups is the predominantly present support for collaboration and execution-related operations (e.g., multistage and distributed execution operations, parameter search, automation, and asset dependency management operations) offered by the state-of-practice subjects.

5 Discussion

We now discuss the results of our study.

Addressing Management Challenges. The supported features found in the subject tools have different implications for the asset management challenges described in Section 2.1.1. Regarding standardized management methods, the lack of uniformity on the type and how the subjects support different features and operations such as asset tracking highlights the lack of standard practices and interoperability. However, tools such as MLFlow encourage standardized methods by supporting models of different flavors to be packaged in standard formats with extended information (e.g., model signature and application context). All the subject tools at a minimum support the tracking of asset metadata, highlighting asset tracking as a core operation for asset management. Related to this is the support for versioning, which most tools support at the "copying assets" level. While this level might be sufficient for small-sized assets, large-sized assets (e.g., datasets and models) require more efficient versioning. To ensure that assets are tracked consistently, the asset collection process is usually associated with the execution of particular experiment runs. For example, an instrumented Python source code file logs assets every time it is executed. Similarly, Guild and DVC offer CLI commands to perform experiments and log associated assets every time a user runs experiments.

We found domain-specific operations tailored to machine learning asset types in those tools that explicitly manage specific asset types. For example, ModelHub [54, 55] 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 of them offer explicit reproducibility operations. Using a purposely designed domain-specific language for machine learning is essential to assist users significantly. Many state-of-practice subjects offer basic collaboration features to discover and share different development assets to improve collaboration. We expect advancement in this aspect to achieve the level of concurrent collaboration similar to practices in traditional software engineering. Furthermore, to improve development collaboration, it is also essential to attain interoperable asset formats through standardized methods. Many tools employ different visualization techniques to compare, analyze, and interpret experimental results. We argue that advancement in relevant research areas (e.g., model explainability [16, 67] and interpretability [17, 31]) should be considered in future tools for improved result interpretation and decision making.

Arbitrary Assets and Metadata. At the very least, all subjects offer support to track static metadata and arbitrary files as asset types during machine learning experiments. We consider this as the primary asset type support for asset tracking. Whereas our subject tools support several asset types as described in our feature model in Fig. 4, they commonly track the static information related to essential assets such as models, datasets, and source code. While some tools support and internally store assets such as datasets, models, and source code, most of our subject tools allow tracking of these assets by referencing. For example, some tools only store the pointer to serialized models, datasets, and metadata, such as location path, version, source, and applied transformation, rather than the actual data object. Also, in scenarios where traditional VCS manages source code, some subject tools track information such as commit hashes and messages as the experiment progresses. A common approach for supporting arbitrary metadata allows to use simple key-value for specifying and tracking any information of interest. While this might lead to flexibility, it can limit the potential gains of using an experiment management tool.

Version Control Systems. When tracking text-based assets—mostly the source code—during machine learning experiments, some tools depend on the versioning information obtained from traditional version control systems (VCSs) such as Git. This approach introduces drawbacks, as it requires users to be disciplined enough to create consistent commits as their experiment evolves. The lack of consistent commits also leads to relevant checkpoints for tracking being missing. In addition, the dependency on traditional VCSs increases tooling complexity for users who lack Git experience and may be a deterring factor for adoption. This approach can still be classified as ad hoc, in the sense of lacking a systematic way to collect and manage machine learning assets effectively [36, 82]. We believe a homogeneous approach to asset management, where a user manages all asset types from a single interface, will be a step towards encouraging the adoption of asset management tools. To achieve such, it would seem reasonable to extend traditional VCSs to a system that can support more machine learning asset types beyond text-based ones. This is the approach of DVC, and we consider it a favorable approach for experiment management tools, especially for users who are familiar with Git.

Automatic Asset Collection. 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 experiment management tools 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 of the tools still only support automatic asset collection for a limited number of popular machine learning frameworks, such as TensorFlow and SciKit Learn. Consequently, the future work sections of several considered state-of-research papers[27, 70, 71, 82] mention the need for support for additional frameworks. Some related work promises to solve issues associated with the intrusive asset collection methods by providing implicit asset collection using instrumented file systems [64] and AST code [28, 29].

Key Supported Operations. Our results show that tracking and exploring tracked assets and their metadata are vital operations across the state-of-practice and state-of-research subject tools. Furthermore, dashboards and visualization to aid the interpretation of tracked assets are mostly supported across all our subject tools. This support indicates the crucial need to provide quick insights on multiple experiment outcomes and the evolution of associated assets: dashboards and visualizations aid a better understanding of the relationships between different assets across multiple experiments. According to our findings, the state-of-practice tools offer more operation varieties for users to manage assets than the state-of-research tools. In contrast, there are few tools with support for collaboration in both state-of-research and state-of-practice. The lacking support for collaboration may reflect the typical way of working, where users independently work on machine learning tasks. This case is especially true in research contexts. However, there is a newer trend of collaboration in machine learning projects, especially in big corporations, and this is evident in state-of-practice tools with collaboration-related operation features.

Reusability. Reproducibility is one of the common objectives of using experiment management tools, and there is a significant presence of such features across our subjects in this study. At the minimum, a subject tool that tracks vital experiment assets and their dependencies and supports the `track` and `retrieve` operations essentially offers reproducibility. In contrast, the operations supported by our subject tools only support the basic reusability of assets. For example, they allow the retrieval and reuse of assets from a previous experiment iteration in another iteration or a completely new experiment. As another example, a stage of an experiment pipeline can reuse intermediates from previously executed experiments; i.e., the execution path of a pipeline skips unmodified stages when reproducing a pipeline. Achieving a higher level of reusability in machine learning experiments can potentially benefit use-cases such as synchronous collaboration and software product line (SPL) engineering [77]. However, this level of reusability is currently challenged by the complexity associated with repurposing already built models and decomposing or merging machine learning models [30]. Also, since every machine learning task requires a different set of learning datasets, tuned hyperparameters, and fit learning algorithms, it becomes complicated to reuse some machine learning assets across various experiments. Reusability of machine learning assets can significantly reduce model development time, enhance asynchronous collaboration in development teams, and motivate assets' evolution use-cases in SPL. We expect to see more tools addressing the reusability challenge of machine learning assets in the future.

6 Threats to Validity

External Validity. The majority of tools surveyed are Python-based, and we identify this as a threat to external validity since it may impact result generalization to other tools. Python will remain the most widely used language in machine learning development, primarily because of its abundant machine-learning-related packages. Consequently, we believe our feature model is valid for most experiment management cases. The subject tools' chosen terminologies vary based on the tools' target groups (e.g., machine learning practitioners, data scientists, researchers) or experiment type (e.g., multi-purpose, machine learning, or deep learning experiment). Consequently, we adopted broad terminologies through multiple analysis iterations per tool to ensure uniformity and generalization across all subjects to enhance external validity.

Internal Validity.

Several threats to internal validity arise from our search strategies for identifying relevant tools. First, in our literature and web searches, our sets of search terms could be incomplete. To make them as complete as possible, we incrementally developed them, augmenting them with additional terms found in our preliminary results (in the case of state-of-practice tools, based on the tools' websites and documentation). We also used the additional information sources of our knowledge and papers found via snowballing to derive terms. Second, in the snowballing strategy, our initial selection of state-of-research publications was based on our own knowledge and assessment of the relevance, which is naturally subjective. Using several complementary search strategies based on literature and web searches, own knowledge, and snowballing considerably mitigates the associated threats to validity.

Since we consider a rapidly evolving technology landscape, where the subject tools and their features are subject to constant changes, we provide the snapshot date of accessed information. Since we manually applied the selection criteria to filter for our final subject state-of-practice tools. One threat to the internal validity might be that the collection and filtering are subjective to individual opinions. In addition, our internet exploration using the Google search engine is prone to varying results based on user, time, and search location—personalized user experience. These issues threaten the ability to reproduce the exact search by other researchers. To mitigate these threats, we relied on additional data sources of grey literature, such as market reports to increase the reliability of our data collection process.

We are limited to available online information for the cloud-based services considered in our work. Consequently, we cannot determine internal details such as the details of their storage systems.

Conclusion and Construct Validity. None of the common threats to conclusion and construct validity provided by Wohlin et al. [85] apply to our study.

7 Related Work

Several surveys and comparisons of tools with asset management support exist. We expect more studies in the future as the discussions on standardized machine learning asset management and applied SE engineering practices in machine learning development deepen.

In a recent survey, Alex et al. mine academic and grey literature to identify 29 engineering best practices for machine learning applications [72]. They conducted a survey to show the level of adoption of the recognized engineering practices among 300+ practitioners. Their findings suggest tech companies have higher adoption rates than non-tech companies. The study also reveals the importance of tracking predictions with model versions and input data, which is often supported by experiment management tools. Similar to our research, Alex et al. compare findings from the research point of view with their related use in practice.

Isdahl et al. [46] survey machine learning platforms' support for reproducibility of empirical results. Several platforms considered in the study fall under the machine-learning experiment management systems—which is also the focus of our study. The authors propose a method to assess machine learning platforms' reproducibility and analyze features that improve their support. Ferenc et al. [27] carry out a comparison of machine learning frameworks' features, investigating support for features that include data versioning, graphical dashboards, model versioning, and machine learning workflow support. Weißgerber et al. [83] study 40 machine learning open-source platforms for support of full-stack machine learning research with open science at its core. The authors develop an open science-centred process model that integrates transparency and openness for machine learning research. The authors found 11 tools and platforms to be most central to the research process, and they further analyze them for resource management and model development capabilities.

Similar to our work, these studies [27, 46, 83] consider tools such as StudioML, MLFlow, Weights & Biases, Polyaxon, Comet.ml, Sacred, Sumatra, and DVC. In contrast to our work, they[27, 46, 83] adopted a more coarse-grained understanding of assets and their management operation. This present work is the first systematic investigation of supported asset types (e.g., differentiating between models and data), which is an essential element of the machine learning domain and has practical implications to users of the considered tools (see the discussion in Section 5). Also, this is a first attempt to compare the support features available in management tools used in practice and those proposed in research.

8 Future Work

For future work, we recommend several directions, based on our survey and our proposed positioning of machine learning asset management as an essential discipline facilitating the development of machine learning-based systems.

First, there is a need for further empirical research, studying the effectiveness, usability, and potential impact of machine learning asset management tools in the diverse scenarios encountered by practitioners—a fundamental research challenge that we explore in a recent *research agenda* paper [41]. We recommend controlled experiments to evaluate the effects of different features (e.g., asset collection) and their realization paradigms (e.g., instrumentation-based vs. CLI-based). Such experiments require a systematic selection of subject tools representing the different available features and paradigms, which is enabled by using our feature model as an underlying taxonomy. An orthogonal direction is to study the effectiveness of the tools for different scenarios, including standard and special ones, e.g., large-scale experiments with enormous datasets, large models, and vast amounts of logs and execution traces. We recommend using controlled experiments, user studies, and action research for investigating the performance of different tools in these usage scenarios.

Second, we suggest use of mining software repository (MSR, [35]) methods to understand and characterize the unique properties of machine learning experiments; more specifically, how the assets of such projects are structured and how they evolve. Recent studies [11, 12, 33, 73, 80] are already exploring the use of MSR to establish empirical results on machine learning-based software projects. Future topics of interest are investigations of the co-evolution of machine learning assets, the asset types commonly managed in version control systems, and file naming conventions.

Third, since this work presents a feature-based survey of tools from the user's perspective, the observed and analyzed features are mainly external and user-visible. While some of our findings give insights into to the mechanisms employed by the studied tools (e.g., non-intrusive vs. intrusive asset tracing), several internal aspects were outside our scope, including the internal design and implementation of the tools. A complementary study could shed light on these aspects.

Fourth, based on our results, researchers can improve the understanding of commonalities and differences between machine-learning experiment management tools and traditional development tools, such as IDEs, VCSs, and artifact hosting platforms. This would pave the way for developing a new generation of software engineering tools with integrated support for machine learning asset management. For example, one question of interest could be: what are the standard features and workflows across the different tool classes? Establishing empirical results based on such comparisons can be used to support design decisions on core features to be incorporated in tools, with more specialized features becoming candidates for tool add-ons.

Finally, researchers and tool builders can develop new tools based on our identified key features, by seeking ways to unify, combine, and expand on these features. Examples of desirable features for tools are ML-specific views and automated recommendations for ML-specific tasks. A particular challenge is to enable the interoperability of available tools, after they all come from different developers and vendors, and a standardized solution for asset management is not available yet. As a step in the this, our recent work presents EMMM [45] as a superset meta-model unifying the conceptual structures found in specific tools, together with their relationships and dependencies. Future work could use this metamodel to support transformations between different tools' asset management formats, rising the level of interopability of available tools.

9 Conclusion

This paper discussed asset management as an essential discipline that offers improved machine-learning-specific asset management and operations to address developmental concerns when building machine-learning-based systems. It also presented a survey of 18 state-of-practice and 12 state-of-research systematically selected tools with management support—experiment management—for machine learning assets, identifying their common and distinguishing features. We performed a feature-based analysis and reported our findings using feature models. We identified four top-level features: supported asset types, collection methods, storage methods, and supported operations. 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 obtained results from experiment executions. Our subjects' 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. We found that commonly supported operations include tracking, exploring, and retrieving assets aimed at experiment reproducibility.

Acknowledgments. Wallenberg Academy Sweden.

We thank the anonymous reviewers for their constructive and valuable feedback.

References

[1] Hadil Abukwaik, Andreas Burger, Berima Kweku Andam, and Thorsten Berger. 2018. Semi-automated feature traceability with embedded annotations. In *ICSME*. 529–533. https://doi.org/10.1109/ICSME.2018.00049

[2] Michele Alberti, Vinaychandran Pondenkandath, Marcel Wursch, Rolf Ingold, and Marcus Liwicki. 2018. DeepDIVA: A highly-functional python framework for reproducible experiments. *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR* 2018-Augus (2018), 423–428. https://doi.org/10.1109/ICFHR-2018.2018.00080 arXiv:1805.00329

[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. 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, 291–300.

[4] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*.

[5] Johan Aronsson, Philip Lu, Daniel Strüber, and Thorsten Berger. 2021. A Maturity Assessment Framework for Conversational {AI} Development Platforms. In *SAC*.

[6] A Arpteg, B Brinne, L Crnkovic-Friis, and J Bosch. 2018. Software Engineering Challenges of Deep Learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 50–59. https://doi.org/10.1109/SEAA.2018.00018

[7] Andrew L Beam, Arjun K Manrai, and Marzyeh Ghassemi. 2020. Challenges to the reproducibility of machine learning models in health care. *Jama* 323, 4 (2020), 305–306.

[8] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. 2017. *Deep learning*. Vol. 1. MIT press Massachusetts, USA:.

[9] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. 2015. What is a feature? A Qualitative study of features in industrial software product lines. In *ACM International Conference Proceeding Series*, Vol. 20-24-July. 16–25. https://doi.org/10.1145/2791060.2791108

[10] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The State of Adoption and the Challenges of Systematic Variability Management in Industry. *EMSE* 25, 3 (2020), 1755–1797.

[11] Aaditya Bhatia, Ellis E Eghan, Manel Grichi, William G Cavanagh, Zhen Ming, Bram Adams, et al. 2022. Towards a Change Taxonomy for Machine Learning Systems. *arXiv preprint arXiv:2203.11365* (2022).

[12] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2021. 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).

[13] Dan Bohus, Sean Andrist, and Mihai Jalobeanu. 2017. Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. 493–494.

[14] Remco R Bouckaert. 2004. Estimating replicability of classifier learning experiments. In *Proceedings of the twenty-first international conference on Machine learning*. 15.

[15] Xavier Bouthillier and Gaël Varoquaux. 2020. *Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020*. Technical Report.

[16] Nadia Burkart and Marco F Huber. 2021. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research* 70 (2021), 245–317.

[17] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[18] Ryan Chard, Zhuozhao Li, Kyle Chard, Logan Ward, Yadu Babuji, Anna Woodard, Steven Tuecke, Ben Blaiszik, Michael J. Franklin, and Ian Foster. 2019. DLHub: Model and data serving for science. In *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium, IPDPS 2019*. Institute of Electrical and Electronics Engineers Inc., 283–292. https://doi.org/10.1109/IPDPS.2019.00038 arXiv:1811.11213

[19] Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 1–12. https://doi.org/10.1145/3313831.3376729

[20] K. Czarnecki and S. Helsen. 2006. Feature-based survey of model transformation approaches. *IBM Syst. J.* 45 (July 2006), 621–645. Issue 3.

[21] Daniel N.R. da Silva, Adolfo Simões, Carlos Cardoso, Douglas E.M. de Oliveira, João N Rittmeyer, Klaus Wehmuth, Hermano Lustosa, Rafael S Pereira, Yania Souto, Luciana E.G. Vignoli, Rebecca Salles, S. C. de Heleno, Artur Ziviani, Eduardo Ogasawara, Flavia C Delicato, Paulo F. de Pires, Hardy Leonardo C.P. da Pinto, Luciano Maia, and Fabio Porto. 2019. A conceptual vision toward the management of machine learning models. In *CEUR Workshop Proceedings*, Vol. 2469. 15–27. http://www.master.iag.usp.br/lab/

[22] G De Michell and Rajesh K Gupta. 1997. Hardware/software co-design. *Proc. IEEE* 85, 3 (1997), 349–365.

[23] Mihail Dimitrov and Ibrahim Osman. 2012. The Impact of Cloud Computing on Organizations in Regard to Cost and Security.

[24] Chris Drummond. 2009. Replicability is not reproducibility: nor is it good science. (2009).

[25] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, et al. 2013. The State of the Art in Language Workbenches. In *SLE*.

[26] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Commun. ACM* 39, 11 (1996), 27–34. https://doi.org/10.1145/240455.240464

[27] Rudolf Ferenc, Tamás Viszkok, Tamás Aladics, Judit Jász, and Péter Hegedűs. 2020. Deep-water framework: The Swiss army knife of humans working with machine learning models. *SoftwareX* 12 (2020), 100551. https://doi.org/10.1016/j.softx.2020.100551

[28] Gharib Gharibi, Vijay Walunj, Rakan Alanazi, Sirisha Rella, and Yugyung Lee. 2019. Automated management of deep learning experiments. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2019). https://doi.org/10.1145/3329486.3329495

[29] Gharib Gharibi, Vijay Walunj, Sirisha Rella, and Yugyung Lee. 2019. 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* March (2019), 28–34. https://doi.org/10.1109/RAISE.2019.00013

[30] Javad Ghofrani, Ehsan Kozegar, Anna Lena Fehlhaber, and Mohammad Divband Soorati. 2019. Applying Product Line Engineering Concepts to Deep Neural Networks. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*. Association for Computing Machinery, 72–77. https://doi.org/10.1145/3336294.3336321

[31] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. 2018. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 80–89.

[32] S Gollapudi. 2016. *Practical Machine Learning*. Packt Publishing.

[33] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. 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*. 431–442.

[34] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. 2009. How do scientists develop and use scientific software?. In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. Ieee, 1–8.

[35] Ahmed E Hassan. 2008. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*. IEEE, 48–57.

[36] C Hill, R Bellamy, T Erickson, and M Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 162–170. https://doi.org/10.1109/VLHCC.2016.7739680

[37] Steffen Hillemacher, Nicolas Jäckel, Christopher Kugler, Philipp Orth, David Schmalzing, and Louis Wachtmeister. 2021. *Artifact-Based Analysis for the Development of Collaborative Embedded Systems*. Springer International Publishing, Cham, 315–331. https://doi.org/10.1007/978-3-030-62136-0_17

[38] Geoff Hulten. 2019. *Building Intelligent Systems*. Springer.

[39] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, Kaoutar El Maghraoui, Anupama Murthi, and Punleuk Oum. 2019. ModelOps: Cloud-based lifecycle management for reliable and trusted AI. In *Proceedings - 2019 IEEE International Conference on Cloud Engineering, IC2E 2019*. 113–120. https://doi.org/10.1109/IC2E.2019.00025

[40] Samuel Idowu, Christer Åhlund, Olov Schelén, Christer Ahlund, and Olov Schelen. 2014. Machine learning in district heating system energy optimization. In *PerCom Workshops*. {IEEE} Computer Society, 224–227. https://doi.org/10.1109/PerComW.2014.6815206

[41] Samuel Idowu, Osman Osman, Daniel Strüber, and Thorsten Berger. 2022. On the Effectiveness of Machine Learning Experiment Management Tools. (2022).

[42] Samuel Idowu, Saguna Saguna, Christer Åhlund, and Olov Schelén. 2015. Forecasting heat load for smart district heating systems: A machine learning approach. In *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014*. Institute of Electrical and Electronics Engineers Inc., 554–559. https://doi.org/10.1109/SmartGridComm.2014.7007705

[43] Samuel Idowu, Saguna Saguna, Christer Ahlund, Olov Schelen, Christer Åhlund, and Olov Schelén. 2016. Applied machine learning: Forecasting heat load in district heating system. *Energy and Buildings* 133 (dec 2016), 478–488. https://doi.org/10.1016/j.enbuild.2016.09.068

[44] Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2021. Asset Management in Machine Learning: A Survey. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 51–60. https://doi.org/10.1109/ICSE-SEIP52600.2021.00014

[45] Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2022. EMMM: A Unified Meta-Model for Tracking Machine Learning Experiments. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. to appear.

[46] Richard Isdahl and Odd Erik Gundersen. 2019. Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms. In *2019 15th International Conference on eScience (eScience)*. IEEE. https://doi.org/10.1109/escience.2019.00017

[47] ISO/IEC/IEEE. 2010. 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)* (2010), 1–51. https://doi.org/10.1109/IEEESTD.2010.5551093

[48] Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. 1990. *Feature-Oriented Domain Analysis {(FODA)} Feasibility Study*. Technical Report. Carnegie-Mellon University, Pittsburgh, PA, USA.

[49] B Kitchenham and S Charters. 2007. Guidelines for performing systematic literature reviews in software engineering. (2007). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471

[50] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas - Proceedings of the 20th International Conference on Electronic Publishing, ELPUB 2016*. 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[51] Fumihiro Kumeno. 2020. Sofware engineering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13, 4 (2020), 463–476. https://doi.org/10.3233/idt-190160

[52] Patrick Lam, Jens Dietrich, and David J. Pearce. 2020. Putting the Semantics into Semantic Versioning. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Virtual, USA) *(Onward! 2020)*. Association for Computing Machinery, New York, NY, USA, 157–179. https://doi.org/10.1145/3426428.3426922

[53] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. 2017. A Classification of Variation Control Systems. In *GPCE*.

[54] Hui Miao, Ang Li, Larry S Davis, and Amol Deshpande. 2016. ModelHub : Lifecycle Management for Deep Learning. *University of Maryland* 0 (2016). arXiv:1611.06224

[55] H Miao, A Li, L S Davis, and A Deshpande. 2017. Towards Unified Data and Lifecycle Management for Deep Learning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 571–582. https://doi.org/10.1109/ICDE.2017.112

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

[57] M Migliore, T M Morse, A P Davison, L Marenco, G M Shepherd, and M L Hines. 2003. Making models publicly accessible to support computational neuroscience. *Neuroinformatics* 1 (2003).

[58] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.

[59] Marçal Mora-Cantallops, Salvador Sánchez-Alonso, Elena García-Barriocanal, and Miguel-Angel Sicilia. 2021. Traceability for trustworthy AI: A review of models and tools. *Big Data and Cognitive Computing* 5, 2 (2021), 20.

[60] Marçal Mora-Cantallops, Salvador Sánchez-Alonso, Elena García-Barriocanal, and Miguel-Angel Sicilia. 2021. Traceability for Trustworthy AI: A Review of Models and Tools. *Big Data and Cognitive Computing* 5, 2 (2021). https://doi.org/10.3390/bdcc5020020

[61] Mohammad Hossein Namaki, Avrilia Floratou, Fotis Psallidas, Subru Krishnan, Ashvin Agrawal, and Yinghui Wu. 2020. Vamsa: Tracking provenance in data science scripts. arXiv:2001.01861

[62] Damir Nešić, Jacob Krüger, Ştefan Stănciulescu, and Thorsten Berger. 2019. Principles of feature modeling. *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (aug 2019), 62–73. https://doi.org/10.1145/3338906.3338974

[63] Damir Nešić, Jacob Krüger, Stefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *FSE*.

[64] Alexandru A Ormenisan, Mahmoud Ismail, Seif Haridi, and Jim Dowling. 2020. Implicit Provenance for Machine Learning Artifacts. *MLSys'20* (2020), 3.

[65] Yang Peili, Yin Xuezhen, Ye Jian, Yang Lingfeng, Zhao Hui, and Liang Jimin. 2018. Deep learning model management for coronary heart disease early warning research. In *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE. https://doi.org/10.1109/icccbda.2018.8386577

[66] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2016. Towards Predicting Feature Defects in Software Product Lines. In *FOSD*.

[67] Ribana Roscher, Bastian Bohn, Marco F Duarte, and Jochen Garcke. 2020. Explainable machine learning for scientific insights and discoveries. *Ieee Access* 8 (2020), 42200–42216.

[68] Jagadeesh Chandra Bose RP, Kapil Singi, Vikrant Kaulgud, Kanchanjot Kaur Phokela, and Sanjay Podder. 2019. Framework for trustworthy software development. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. IEEE, 45–48.

[69] Iqbal H Sarker, Faisal Faruque, Ujjal Hossen, and Atikur Rahman. 2015. A Survey of Software Development Process Models in Software Engineering. *International Journal of Software Engineering and Its Applications* 9, 11 (2015), 55–70. https://doi.org/10.14257/ijseia.2015.9.11.05

[70] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. [n.d.]. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. ([n. d.]), 8.

[71] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2018. Declarative Metadata Management: A Missing Piece in End-To-End Machine Learning. *SysML 2018* (2018), 3.

[72] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. 2020. Adoption and Effects of Software Engineering Best Practices in Machine Learning. *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Oct 2020). https://doi.org/10.1145/3382494.3410681

[73] Andrew J Simmons, Scott Barnett, Jessica Rivera-Villicana, Akshat Bajaj, and Rajesh Vasa. 2020. 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)*. 1–11.

[74] Renan Souza, Liliane Neves, Leonardo Azeredo, Ricardo Luiz, Elaine Tady, Paulo Cavalin, and Marta Mattoso. 2018. Towards a human-in-the-loop library for tracking hyperparameter tuning in deep learning development. In *CEUR Workshop Proceedings*, Vol. 2170. 84–87. https://github.com/hpcdb/DL-Steer

[75] Vinay Sridhar, Sriram Subramanian, Dulcardo Arteaga, Swaminathan Sundararaman, Drew Roselli, and Nisha Talagala. 2018. Model governance: Reducing the anarchy of production ML. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 351–358. https://www.usenix.org/conference/atc18/presentation/sridhar

[76] Stefan Strüder, Mukelabai Mukelabai, Daniel Strüber, and Thorsten Berger. 2020. Feature-Oriented Defect Prediction. In *SPLC*.

[77] Vijayan Sugumaran, Sooyong Park, and Kyo C Kang. 2006. Software product line engineering. *Commun. ACM* 49, 12 (2006), 28–32.

[78] Rachael Tatman, Jake Vanderplas, and Sohier Dane. 2018. A Practical Taxonomy of Reproducibility for Machine Learning Research. *Reproducibility in ML Workshop, ICML'18* Ml (2018). https://hub.docker.com/r/pangwei/tf1.1/

[79] J Tsay, T Mummert, N Bobroff, A Braz, and P Westerink. 2018. Runway: machine learning model experiment management tool. *SysML* February (2018), 1–3. https://doi.org/10.1145/3097983.3098021

[80] Bart van Oort, Luís Cruz, Maurício Aniche, and Arie van Deursen. 2021. The Prevalence of Code Smells in Machine Learning projects. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 1–8.

[81] Joaquin Vanschoren, Jan N van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: Networked Science in Machine Learning. *SIGKDD Explor. Newsl.* 15, 2 (jun 2014), 49–60.

[82] Manasi Vartak, Harihar Subramanyam, Wei-En En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: a system for machine learning model management. In *the Workshop*. ACM Press, 1–3. https://doi.org/10.1145/2939502.2939516

[83] Thomas Weißgerber and Michael Granitzer. 2019. Mapping platforms into a new open science model for machine learning. *it - Information Technology* 61, 4 (2019), 197–208. https://doi.org/10.1515/itit-2018-0022

[84] Rüdiger Wirth. 2000. 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* 24959 (2000), 29–39. http://dx.doi.org/10.1.1.198.5133

[85] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Vol. 9783642290. Springer-Verlag Berlin Heidelberg. 1–236 pages. https://doi.org/10.1007/978-3-642-29044-2

[86] Justin M. Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T. Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, Cristina Garcia Cardona, Brian Van Essen, and Matthew Baughman. 2018. CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. *BMC Bioinformatics* 19, S18 (dec 2018), 491. https://doi.org/10.1186/s12859-018-2508-4

[87] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. 2016. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research* 4, 1 (2016), 23–45.

[88] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya Parameswaran. 2018. Accelerating Human-in-the-Loop Machine Learning: Challenges and Opportunities. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning* (Houston, TX, USA) *(DEEM'18)*. Association for Computing Machinery, New York, NY, USA, Article 9, 4 pages. https://doi.org/10.1145/3209889.3209897

[89] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.