Management of Machine Learning Lifecycle Artifacts: A Survey

Marius Schlegel
TU Ilmenau, Germany
marius.schlegel@tu-ilmenau.de

Kai-Uwe Sattler
TU Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

The explorative and iterative nature of developing and operating ML applications leads to a variety of artifacts, such as datasets, features, models, hyperparameters, metrics, software, configurations, and logs. In order to enable comparability, reproducibility, and traceability of these artifacts across the ML lifecycle steps and iterations, systems and tools have been developed to support their collection, storage, and management. It is often not obvious what precise functional scope such systems offer so that the comparison and the estimation of synergy effects between candidates are quite challenging. In this paper, we aim to give an overview of systems and platforms which support the management of ML lifecycle artifacts. Based on a systematic literature review, we derive assessment criteria and apply them to a representative selection of more than 60 systems and platforms.

1. INTRODUCTION

Machine learning (ML) approaches are well established in a wide range of application domains. In contrast to engineering traditional software, the development of ML systems is different: data and feature preparation, model development, and model operation tasks are integrated into a unified lifecycle which is often iterated several times [26, 9, 53]. Although there are systems and tools that provide support for a broad range of tasks within the ML lifecycle, such as data cleaning and labeling, feature engineering, model design and training, experiment management, hyperparameter optimization, and orchestration, achieving comparability, traceability, and reproducibility of model and data artifacts across all lifecycle steps and iterations is still challenging.

To meet these requirements, it is necessary to capture the input and output artifacts of each lifecycle step and iteration. That includes model artifacts and data-related artifacts, such as datasets, labels, and features. Reproducibility also requires capturing software-related artifacts, such as code, configurations, and environmental dependencies. By additionally considering metadata, such as model parameters, hyperparameters, quality metrics, and execution statistics, comparability of artifacts is enabled.

Since the manual management of ML artifacts is simply not efficient, systems and platforms provide support for the systematic collection, storage, and management of ML lifecycle artifacts, which we collectively referred to as *ML artifact management systems* (ML AMSs)[1]. Since ML AMSs are often integrated into general ML development platforms or frameworks for a subset of the ML lifecycle tasks, it is typically not obvious what the precise functional and non-functional scope of an AMS is, how an AMS compares to others, and to what extent possible synergy effects can be exploited through tool-chaining.

The objective of this paper is to provide a comprehensive overview of AMSs from academia and industry. We address the following research questions: (RQ1) What are criteria to describe, assess, and compare AMSs? (RQ2) Which AMSs exist in academia and industry, and what are their functional and non-functional properties according to the assessment criteria? To answer these questions, we conducted a systematic literature review.

The paper is organized as follows: § 2 gives an overview of related work. § 3 describes the ML lifecycle and concretizes the tasks of ML lifecycle management. Based on the conducted systematic literature review, § 4 discusses criteria for assessing AMSs w. r. t. their functional and non-functional scope of features. § 5 applies the criteria to the 64 identified AMSs and discusses the results.

2. RELATED WORK

In recent years, both academia and industry have produced a variety of systems that provide artifact collection and management support for individual steps of ML lifecycles [101, 130, 3, 25, 63, 28, 48, 18, 50]. Authors often compare with related works in the scope of the particular system, which, however,

[1]Whenever we use just "AMS", we refer to an ML AMS.

does not enable the comparability with a broad range of systems and criteria.

This problem has been addressed by a few surveys [69, 168, 67]. In the context of reproducibility of empirical results, Isdahl et al. [69] have investigated what support is provided by existing experiment management systems. However, these systems cover only a subset of the ML lifecycle. Weißgerber et al. [168] develop an open-science-centered process model for ML research as a common ground and investigate 40 ML platforms and tools. However, the authors analyze only 11 platforms w. r. t. ML workflow support capabilities and their properties.

In contrast to the aforementioned studies and surveys, Idowu et al. [67] adopt a more fine-grained understanding of artifacts and system capabilities which is most closely to our work. Based on a selection of 17 experiment management systems and tools, the authors develop a feature model for assessing their capabilities. Although this survey shows parallels to our work, the authors only consider a limited selection of systems which is, again, only focused on the area of experiment tracking and management.

3. MACHINE LEARNING LIFECYCLE ARTIFACT MANAGEMENT

In this section, we discuss the steps of ML lifecycles based on typical ML workflows (§ 3.1), derive the tasks of ML artifact management, and outline the support ML AMSs should provide (§ 3.2).

3.1 ML Lifecycle

In contrast to traditional software engineering, the development of ML-powered applications is more iterative and explorative. Thus, developers have adapted their processes and practices for ML: Following methodologies in the context of data science, data analytics and data mining, such as TDSP [102], KDD [45], CRISP-DM [137, 169], or ASUM-DM [64], workflows specialized for ML have been established [26, 9, 53, 128]. Despite minor differences, ML workflows contain both data-centric and model-centric steps and often multiple feedback loops among the different stages, which leads to a *lifecycle*. Fig. 1 depicts a common view on the ML lifecycle.

The ML lifecycle consists of four stages: Requirements Stage, Data-oriented Stage, Model-oriented Stage, and Operations Stage. Starting with the Requirements Stage, the requirements for the model to be developed are derived based on the application requirements [163]. This stage is dedicated to three major decisions: (1.) to decide which functionality and interfaces to realize, (2.) to decide which types of models are best suited for the given problem, and

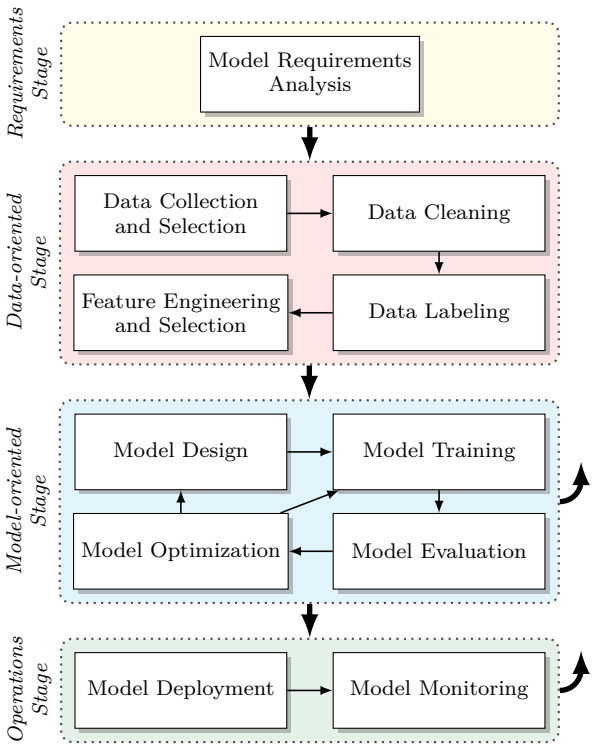

Figure 1: Typical ML lifecycle.

(3.) to decide which types of data to work on.

The Data-oriented Stage starts with the Data Collection and Selection step. Datasets, either internal or publicly available, are searched, or individual ones are collected and the data most suitable for the subsequent steps is selected (e. g. dependent on data quality, bias, etc.). By using available generic datasets, models may be (pre-)trained (e. g. ImageNet for object recognition), and later, by using transfer learning [24, 111] along with more specific data, trained to a more specific model. Then, in the Data Cleaning step, datasets are prepared, removing inaccurate or noisy records. As required by most of the supervised learning techniques to be able to induce a model, data labeling is used to assign a ground truth label to each dataset record. Subsequently, feature engineering and selection is performed to extract and select features for ML models. For some models, such as convolutional neural networks, this step is directly intertwined with model training.

The Model-oriented Stage starts with the Model Design step. Often, existing model designs and neural network architectures are used and tailored towards specific requirements. During model training, the selected models are trained on the collected and preprocessed datasets using the selected features and their respective labels. Subsequently, in the Model Evaluation step, developers evaluate a trained model on test datasets using predefined metrics, such as ac-

curacy or F1-score. In critical application domains, this step also includes extensive human evaluation. The subsequent Model Optimization step is used to fine-tune the model, especially its hyperparameters. In the context of the model development steps, we refer to an experiment as a sequence of model development activities that result in a trained model but do not include cycles to previous steps.

Finally, in the Operations Stage, the model is distributed to the target systems and devices, either as an on-demand (online) service or in batch (offline) mode (Model Deployment), as well as continuously monitored for metrics and errors during execution and use (Model Monitoring). In particular, CI/CD practices from software engineering are adapted.

As illustrated by Fig. 1, multiple feedback loops from steps of the Model-oriented Stage or the Operations Stage to any step before may be triggered by insufficient accuracy or new data. Moreover, Sculley et al. [131] point out, that the model development often takes only a fraction of the time required to complete ML projects. Usually, a large amount of tooling and infrastructure is required to support data extract, transform, and load (ETL) pipelines, efficient training and inference, reproducible experiments, versioning of datasets and models, model analysis, and model monitoring at scale. The creation and management of services and tools can ultimately account for a large portion of the workload of ML engineers, researchers, and data scientists.

3.2 Management of ML Lifecycle Artifacts

Within the steps of the ML lifecycle, a variety of artifacts is created, used, and modified: Datasets, labels and annotations, and feature sets are inputs and outputs of steps in the Data-oriented Stage. Moreover, data processing source code, logs, and environmental dependencies are created and/or used. In the Model-oriented Stage, results from the Data-oriented Stage are used to develop and train models. In addition, metadata such as parameters, hyperparameters, and captured metrics as well as model processing source code, logs, and environment dependencies are artifacts that are created and/or used in this stage. The Operations Stage requires trained models and corresponding dependencies such as libraries and runtimes (e. g. via Docker container), uses model deployment and monitoring source code which is typically wrapped into a web service with a REST API for on-demand (online) service or scheduled for batch (offline) execution, and captures execution logs and statistics. To achieve comparability, traceability, and reproducibility of produced data and model artifacts across multiple lifecycle iterations, it is essential to also capture metadata artifacts that can be easily inspected afterwards (e. g. model parameters, hyperparameters, lineage traces, performance metrics) as well as software artifacts.

Manual management of artifacts is simply not efficient due to the complexity and the required time. To meet the above requirements, it is necessary to systematically capture any input and output artifacts and to provide them via appropriate interfaces. *ML artifact management* includes any methods and tools for managing ML artifacts that are created and used in the development, deployment, and operation of ML-based systems. Systems supporting ML artifact management, collectively referred to as *ML artifact management systems* (ML AMSs), provide the functionality and interfaces to adequately record, store, and manage ML lifecycle artifacts.

4. ASSESSMENT CRITERIA

The goal of this section is to define criteria for the description and assessment of AMSs. Based on a priori assumptions, we first list functional and non-functional requirements. We then conduct a systematic literature review according to Kitchenham et al. [81]: Using well-defined keywords, we search ACM DL, DBLP, IEEE Xplore, and SpringerLink for academic publications as well as Google and Google Scholar for web pages, articles, white papers, technical reports, reference lists, source code repositories, and documentations. Next, we perform the publication selection based on the relevance for answering our research questions. To avoid overlooking relevant literature, we perform one iteration of *backward snowballing* [170]. Finally, we iteratively extract assessment criteria and subcriteria, criteria categories, as well as the functional and non-functional properties of concrete systems and platforms based on concept matrices. The results are shown in Tab. 1, which outlines categories, criteria (italicized), subcriteria (in square brackets).

Lifecycle Integration.

This category describes for which parts of the ML lifecycle a system provides artifact collection and management capabilities. The four stages form the criteria, with the steps assigned to each stage forming the subcriteria (cf. § 3.1).

Artifact Support.

Orthogonal to the previous category, this category indicates which types of artifacts are supported and managed by an AMS. Based on the discussion in § 3.2, we distinguish between the criteria *Data-related*, *Model*, *Metadata*, and *Software Artifacts*.

The criteria *Data-related Artifacts* and *Model Ar-*

Category	Criteria and Subcriteria
Lifecycle Integration	*Requirements Stage* [Model Requirements Analysis] *Data-oriented Stage* [Data Collection & Selection, Data Preparation & Cleaning, Data Labeling, Feature Engineering & Selection] *Model-oriented Stage* [Model Design, Model Training, Model Evaluation, Model Optimization] *Operations Stage* [Model Deployment, Model Monitoring]
Artifact Support	*Data-related Artifacts* [Dataset, Annotations & Labels, Features] *Model Artifacts* [Model] *Metadata Artifacts* [Identification, Model Parameters, Model Hyperparameters, Model Metrics, Experiments & Projects, Pipelines, Execution Logs & Statistics] *Software Artifacts* [Source Code, Notebooks, Configurations, Environment]
Operations	*Logging & Versioning* [Log/Capture, Commit, Revert/Rollback] *Exploration* [Query, Compare, Lineage, Provenance, Visualize] *Management* [Modify, Delete, Execute & Run, Deploy] *Collaboration* [Export & Import, Share]
Collection & Storage	*Collection Automation* [Intrusive, Non-intrusive] *Storage Type* [Filesystem, Database, Object/BLOB Storage, Repository] *Versioning* [Repository, Snapshot]
Interfaces & Integration	*Interface* [API/SDK, CLI, Web UI] *Language Support & Integration* [Languages, Frameworks, Notebook]
Operation & Licensing	*Operation* [Local, On-premise, Cloud] *License* [Free, Non-free]

Table 1: Assessment categories, criteria, and subcriteria.

tifacts represent core resources that are either input, output, or both for a lifecycle step. *Data-related Artifacts* are datasets (used for training, validation, and testing), annotations and labels, and features (cf. corresponding subcriteria). *Model Artifacts* are represented by trained models (subcriterion *Model*).

The criteria *Metadata Artifacts* and *Software Artifacts* represent the corresponding artifact types, that enable the reproducibility and traceability of individual ML lifecycle steps and their results. The criterion *Metadata Artifacts* covers different types of metadata: (i) identification metadata (e. g. identifier, name, type of dataset or model, association with groups, experiments, pipelines, etc.); (ii) data-related metadata; (iii) model-related metadata, such as inspectable model parameters (e. g. weights and biases), model hyperparameters (e. g. number of hidden layers, learning rate, batch size, or dropout), and model quality & performance metrics (e. g. accuracy, F1-score, or AUC score); (iv) experiments and projects, which are abstractions to capture data processing or model training runs and to group related artifacts in a reproducible and comparable way; (v) pipelines, which are abstractions to execute entire ML workflows in an automated fashion and relates the input and output artifacts required per step as well as the glue code required for processing; (vi) execution-related logs & statistics.

The criterion *Software Artifacts* comprises source code and notebooks, e. g. for data processing, experimentation and model training, and serving, as well as configurations and execution-related environment dependencies and containers, e. g. Conda environments, Docker containers, or virtual machines.

Operations.

This category indicates the operations provided by an AMS for handling and managing ML artifacts. It comprises the criteria *Logging & Versioning*, *Exploration*, *Management*, and *Collaboration*.

The criterion *Logging & Versioning* represents any operations that enable logging or capturing single artifacts (subcriterion *Log/Capture*), creating checkpoints of a project or an experiment comprising several artifacts (subcriterion *Commit*), and reverting or rolling back to an earlier committed or snapshot version (subcriterion *Revert/Rollback*).

The criterion *Exploration* includes any operations that help to gain concrete insights into the results of data processing pipelines, experiments, model training results, or monitoring statistics. These operations are differentiated by the subcriteria *Query*, *Compare*, *Lineage*, *Provenance*, and *Visualize*. *Query* operations may be represented by simple searching and listing functionality, more advanced filtering functionality (e. g. based on model performance metrics), or a comprehensive query language. *Compare* indicates the presence of operations for the comparison between two or more versions of artifacts. In terms of model artifacts, this operation may be used to select the most promising model from a set of can-

didates (*model selection*), either in model training and development [122] or in model serving (e. g. best performing predictor) [33]. *Lineage* represents any operations for tracing the lineage of artifacts, i. e. which input artifacts led to which output artifacts, and thus provide information about the history of a model, dataset, or project. *Provenance* represents any operations, which in addition provide information about which concrete transformations and processes converted inputs into an output. *Visualize* indicates the presence of functionality for graphical representation of model architectures, pipelines, model metrics, or experimentation results.

The criterion *Management* characterizes operations for handling and using stored artifacts. The subcriteria *Modify* and *Delete* indicate operations for modifying or deleting logged and already stored artifacts. *Execute & Run* comprises operations that are interfaces for the execution and orchestration of data processing or model training experiments and pipelines. The subcriterion *Deploy* refers to deployment operations for offline (batch) and online (on-demand) model serving.

The criterion *Collaboration* indicates the presence of operations for collaboration which enable simple export/import functionality (subcriterion *Export & Import*) as well as sharing of artifacts among internal company and team members or publishing of artifacts for external instances (subcriterion *Share*).

Collection & Storage.

This category describes the model for artifact collection and storage based on the criteria *Collection Automation*, *Storage Type*, and *Versioning*.

The criterion *Collection Automation* represents the degree of manual effort required to collect and capture ML artifacts. The collection of artifacts is either intrusive, which requires engineers to explicitly add instructions or API calls within the source code, non-intrusive, which means that no explicit manual actions are required and the collection is performed automatically, or both.

Storage Type describes the type of storage used and supported by an AMS. We identified the subcriteria *Filesystem*, *Database*, *Object/BLOB Storage* and *Repository*. An AMS can support multiple types of storage, and also hybrid variants are possible.

While local filesystems are often the means of choice to store artifacts for small and manageable experiments, such as smaller textual datasets (e. g. in `.csv` or `.parqet` files) or metadata (e. g. in `.yaml` files), distributed file systems (e. g. HDFS) are rather used for larger projects and permanently scalable solutions. These are suitable for both large numbers and large files, as is the case for image, video, and text datasets as well as trained models in serialized file formats (e. g. `.pb`, `.onnx`, `.pkl`, `.pt`, or `.pmml`). Often, these are also stored on separate storage servers or clusters, and provided with an API to fulfill availability and replication requirements.

Databases are established for storing a wide range of different types of data. In the context of artifact management, large tabular, sequential, and text datasets may be stored in (object-)relational or time-series databases, metadata in relational databases, and logs in key-value databases. While modern and widely used DBMSs (e. g. PostgreSQL [120]) provide BLOB data types, these often have limitations regarding maximum file sizes, which is why also object/BLOB stores are often used.

Repositories are version-controlled and typically suited best for source code and text. It is also possible to add version control on top of other storage types, so that the storage of large files in a distributed file system is combined with version control. For example, Git LFS (Large File Storage) replaces large files such as image, audio, and video datasets, with text pointers inside Git, while storing the file contents on a remote file server. This preserves and accelerates typical workflows, while enabling versioning of large amounts of data or large files.

The criterion *Versioning* characterizes the way versioning of artifacts is done. Either versioning is delegated to a traditional version control system (e. g. Git or Mercurial) and managed by means of a repository (see also corresponding storage type). A repository contains several to all artifacts associated with a project. In contrast, versioning may be done by following a snapshot-based approach: Snapshots are created manually for each individual artifact and possibly independently of other artifacts.

Interfaces & Integration.

This category characterizes an AMS's user interfaces and integration capabilities. The criterion *Interface* states the type of provided interfaces that may be based on an API (e. g. a REST interface) or a higher-level SDK, based on a command line interface (CLI), or based on a web application. *Language Support & Integration* distinguishes between the integration into programming languages (e. g. into Python via provided libraries), integration into well-known frameworks providing functional integration for the steps of the ML lifecycle (e. g. data processing with Apache Spark [13], model training with TensorFlow [56], or model orchestration with Metaflow [110]) and notebook support (e. g. Jupyter [121], Apache Zeppelin [12], or TensorBoard [55])

Operation & Licensing.

The last category covers two non-functional, usage-related criteria *Operation* and *License*. The criterion *Operation* defines whether the operation of a system or tool is local (e. g. the case for Python libraries, subcriterion *Local*), on-premise (e. g. the case for server-based systems, subcriterion *On-premise*) or by a dedicated cloud provider (e. g. for hosted services, subcriterion *Cloud*). The criterion *License* describes the type of license, which is either classified as free (e. g. public domain, permissive, or copyleft licenses) or non-free (e. g. non-commercial or proprietary licenses), and which may be further concretized by the concrete license.

5. ASSESSMENT

This section presents the assessment of concrete ML AMSs. As part of the systematic literature review (see § 4), we identified a total of 64 systems and platforms from research and industry. We assessed these based on our criteria and subcriteria (cf. § 4). An additional result of this assessment is the derivation of the classes which aim to group systems with high similarity regarding lifecycle integration, artifact support, and functionality. Fig. 2 visualizes the results at criteria level (depending on a criterion's semantic, we consider either its fulfillment or presence) and the AMS classes. The following two subsections discuss the assessment results on these orthogonal dimensions: Based on the derived classes, we first provide an overview of the identified AMSs and their core characteristics (§ 5.1). Subsequently, we discuss to what extent the criteria and subcriteria are fulfilled by the systems within their classes (§ 5.2).

5.1 Discussion Along Classes

A general observation is that the focus of ML-supporting systems is often not obvious: The boundaries are blurred between systems that purely provide functionality for the development of ML-based systems to those that purely focus on the management, storage, and deployment of ML artifacts (and typically complement the former). Therefore, we classified the assessed systems based on the characteristics of the criteria within the categories Lifecycle Integration, Artifact Support, and Operations into five classes: Lifecycle Management, Pipeline Management, Experiment Management, Model Management, and Dataset & Feature Management.

Lifecycle Management.

This class includes systems and platforms that focus on the entire ML lifecycle and, typically beyond broad functional support for the different tasks within the ML lifecycle, provide artifact management capabilities for nearly all ML lifecycle steps.

Microsoft *Azure ML* [103], Amazon *SageMaker* [130, 8], Google *Vertex AI* [57], IBM *Watson Studio* [63, 123, 65], *Comet* [31], *DataRobot AI Cloud Platform* [142, 36], and *Cloudera Data Science Workbench* [35] are "all-in-one" ML as a service (MLaaS) platforms which are comparable in objective and functional scope. Although the direct integration into a provider's cloud infrastructure usually offers rich processing possibilities, such as the scaling of computing and memory resources, the usage is subject to the fees and pricing of the provider. Furthermore, the usage of MLaaS may not be possible in certain application scenarios due to data protection requirements and legal regulations.

In contrast, open-source AMSs, such as *MLflow* [174, 28, 83, 84], *ClearML* [7, 6], *Polyaxon* [118, 119], and *Hopsworks* [70, 113, 112, 88, 89], can be deployed both in the cloud and on-premise. MLflow focuses on capturing, storing, managing, and deploying ML artifacts: MLflow Tracking is an API for logging experiment runs, including code and data dependencies, via automatic or manual instrumenting application code. These runs can be viewed, compared, and search through an API or the UI. MLflow Models are a convention for packaging models and their dependencies, that is compatible with diverse serving environments. MLflow Projects provides a standard format for packaging reusable and reproducible project code. MLflow Model Registry is a hub for storing models and managing their deployment lifecycles. ClearML and Polyaxon also aim at simplifying the entire ML lifecycle. Both provide a comparable range of functions, plus model monitoring and resource management. The Hopsworks platform also has a comparable range of functions but additionally includes big data and GPU support for highly scalable learning, HDFS extended by a distributed hierarchical metadata service using a NewSQL database (*HopsFS*), Github-like project management, and an integrated feature store.

Valohai [153] is a platform for managing and versioning ML pipelines from data extraction to model deployment. Its objective is comparable to the previous AMSs. The three layers of the platform, (Web) Application Layer, Computer Layer, and Data Layer, can be flexibly deployed in the cloud (e. g. Valohai or own AWS account) or on-premise.

In relation to the previously described AMSs, *DVC* [18, 74, 73] is a complementary ML artifact management tool for ML pipelines. DVC's versioning is built upon Git and provides experiment branching semantics, push and pull processing of

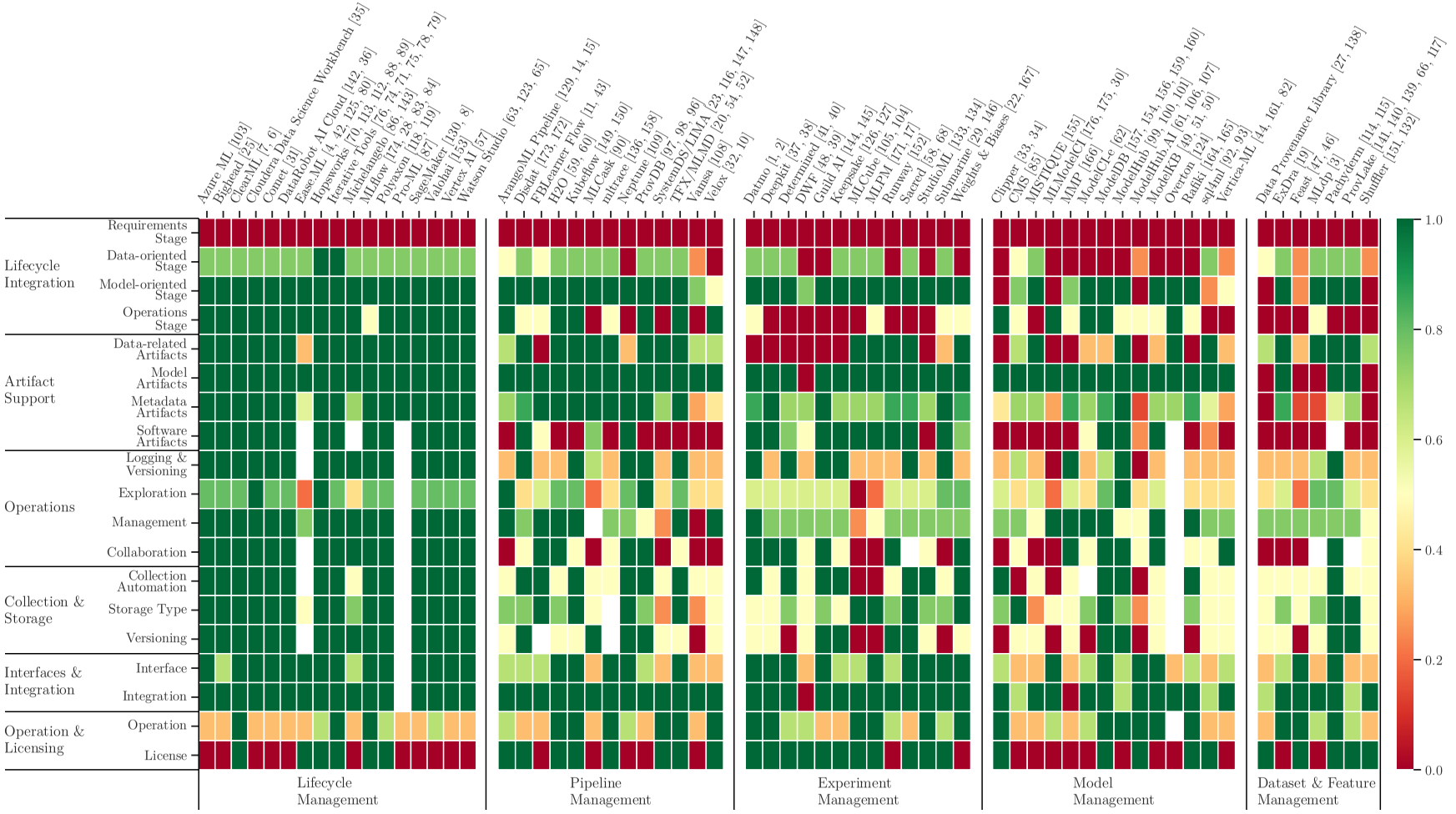

Figure 2: The heatmap visualizes the assessment results. For each criterion, the number of subcriteria fulfilled/present was determined, related to the total number of subcriteria, and normalized to the value range $[0, 1]$. The degree of fulfillment/presence of a criterion (y-axis) by the investigated systems (x-axis) is represented by the hue of a cell ranging from dark red (i. e. not fulfilled/present) to dark green (i. e. completely fulfilled/present). If the fulfillment/presence of all subcriteria of a criterion is unclear or not exactly known, the corresponding cell's hue is white. The last criterion "License" is an exception due to its binary character: the hue is either dark red ("non-free") or dark green ("free").

bundles of models, data, and code, as well as automatic metric tracking. Recently, the company behind, Iterative, built a tool ecosystem around DVC to achieve ML lifecycle management [76]: a library for implementing CI/CD in ML projects (CML) [71, 72], a library for logging ML metrics and metadata (DVCLive) [75], a web application for seamless data and model management, experiment tracking, visualization, and collaboration (Studio) [78, 77], a model registry and deployment tool (MLEM) [79].

In contrast, *Ease.ML* is an ML lifecycle management system specifically targeting non-ML experts [4, 42, 125, 80]. Built on top of existing data ecosystems and techniques, Ease.ML guides users step-by-step: Starting by automatic data ingestion and augmentation, automatic feasibility studies, data noise debugging, data acquisition, scalable multi-tenant automatic training, continuous integration, and ending with continuous quality optimization.

Additionally, we identified proprietary and internally deployed ML lifecycle management platforms from the major tech companies Airbnb, Uber, and LinkedIn. *Bighead* is Airbnb's framework-agnostic ML platform tailored to their use cases and environment [25]. It includes a feature management framework based on the lambda architecture principle [94, 95], a model development and execution management toolkit, a lifecycle management service, an offline training and inference engine, an online inference service with containerization and cloud native architecture, and a container management tool.

Michelangelo is Uber's ML platform which enables internal teams to build, deploy, and operate uniform and reproducible ML pipelines for applications in a microservices-based production environment [86]. Michelangelo consists of a mix of open-source systems and components built in-house, such as a centralized feature store, a domain-specific language (DSL) for feature selection and transformation, the distributed deep learning framework *Horovod* [135] and the model management system *Gallery* [143].

LinkedIn's *Pro-ML* system specifically aims to meet internal scalability requirements [87]. Pro-ML supports a key set of ML lifecycle steps: data exploration and model authoring with an own DSL for feature and model representations and a central feature marketplace, real time and batch model training, model deployment, and model monitoring.

Pipeline Management.

ML pipelines are abstractions to enable a holistic view on data processing, model development, and model deployment workflows. This class includes any AMSs that support the management of ML pipelines and related artifacts. In contrast to the previous class, these systems typically do not comprise either support for the Operations Stage (in a few cases), the management of software artifacts, or both.

Although the pipeline management systems *Velox* [32, 10], *Vamsa* [108], and *ArangoML Pipeline* [129, 14, 15] differ to some extent in their goals, design principles, and intended uses, they overall provide a comparable basic range of artifact management capabilities, with a focus on models and model metadata. Velox provides management of training pipeline orchestration for a set of predeclared models, model performance evaluation, retraining as necessary, and low-latency model inference. Vamsa is a tool for automated provenance tracking of ML pipelines based on static analyses of Python programs. ArangoML Pipeline is an artifact and metadata storage layer for ML pipeline lineage tracking, auditing, reproducibility, and monitoring built around ArangoDB.

In comparison, Apache *SystemDS* [23, 147, 148] is a declarative ML pipeline system. It provides a DSL with declarative language abstractions for different ML lifecycle tasks. High-level scripts are compiled into hybrid execution plans of local, in-memory CPU and GPU operations, as well as distributed operations on Spark based on a tensor data model. Based on the LIMA framework, SystemDS provides fine-grained, multi-level lineage tracing as well as compiler-assisted, full and partial reuse during runtime removing redundancy at different levels of hierarchically composed ML pipelines [116].

Mltrace [136, 158] and *ProvDB* [97, 98, 96] also have a strong focus on lineage and provenance tracing. Mltrace is a Python tool for lineage and tracing of artifacts in ML pipelines. It integrates into existing codebases without requiring redesigning pipelines or rewriting pipeline code. *ProvDB* is a unified provenance, model lineage, and metadata management system founded on a graph-based provenance representation model that generalizes the W3C PROV data model. It features the Neo4j Cypher and Apache Gremlin graph query languages and two query operators for graph segmentation and summarization geared towards the characteristics of provenance information.

Furthermore, the *TensorFlow Extended (TFX)* framework provides libraries for creating ML pipelines for Data-oriented, Model-oriented and Operations Stages, as well as the *ML Metadata (MLMD)* library for metadata management and version control [20, 54, 52]. Embedded in this technical ecosystem, *Kubeflow* enables coordinated deployments of workflows on Kubernetes clusters [149, 150].

Although providing a platform for distributed in-

memory ML and predictive analysis on big data, *H2O*'s [59, 60] ML lifecycle integration and ML artifact management capabilities are mostly comparable. Additionally, H2O aims at easy productionalization of ML models in enterprise environments.

Neptune [109], *FBLearner Flow & Predictor* [11, 43], *MLCask* [90], and *Disdat* [173, 172] additionally support software artifacts and further promote reproducibility. Neptune provides dataset, model, and metadata artifact storage for pipelines, experiment tracking, and workspaces for coordination of projects and collaborating users. FBLearner Flow & Predictor is Facebook's proprietary ML pipeline development and processing system. It comprises Flow, a DAG-based pipeline management system that facilitates experimentation, training, and comparison of models, and Predictor, an online inference framework based on an own model format. MLCask is a pipeline-oriented AMS. It builds upon Git-inspired versioning of pipeline components with non-linear version control semantics for collaborative environments. Disdat manages ML pipelines and related ML artifacts by building upon two core abstractions: bundles and contexts. A bundle is a versioned, typed, immutable collection of data and files. A context is a view abstraction gathering a sharable set of bundles and assisting with managing bundles across multiple locations such as local and cloud storage environments.

Experiment Management.

Systems and platforms of this class aim to achieve comparability and reproducibility of exploratory ML experiments for model development, training, and optimization. They typically complement model training frameworks and AMSs of the previous class, since the results often serve as a starting point for subsequent pipeline creation and execution. Because of this, there is typically no or only limited support for the Model Operations Stage, but support for metadata and especially software artifacts.

The *Deep-water Framework (DWF)* [48, 39] has a basic set of functionality that enables tracking of experiments and training runs, involved artifacts' identifiers, as well as configurations and performance metrics. DWF supports only a predefined set of models provided by TensorFlow and scikit-learn and no storage or versioning of trained models. *StudioML* [133, 134] additionally captures model artifacts without the necessity of modifying experiment code.

Focusing on reproducibility for the Model Development and Model Operations Stages, *MLCube* [105, 104] and *MLPM* [171, 17] both provide model packaging capabilities: MLCube is a library for packaging ML tasks and models, which enables sharing and consistent reproduction of models, experiments, and benchmarks. MLPM enables users to adopt existing ML algorithms and libraries, resolving dependencies, and deploying as HTTP services.

An even higher degree of reproducibility is provided by *Guild AI* [144, 145], *Datmo* [1, 2], *Deepkit* [37, 38], and *Keepsake* [126, 127], which additionally capture any necessary software including source code, dependencies, execution environment, and logs. *Runway* [152], *Sacred* [58, 68], and *Weights & Biases (W&B)* [22, 167] furthermore provide capabilities for the management of data-related artifacts.

Apache Submarine [29, 146] and *Determined* [41, 40] both provide functional interfaces and integration for popular ML training, experimentation, artifact management frameworks (e. g. TensorFlow, PyTorch, MLflow, and TensorBoard) and Python SDKs for different stages of model development without requiring extra infrastructure knowledge for orchestration. Submarine supports both on-premise clusters managed by Kubernetes or YARN, and clouds to ensure portability and resource-efficiency.

Model Management.

AMSs of this class treat models and model metadata as central abstractions. While typically limited integration with the Data-oriented Stage and capabilities for managing data-related artifacts are provided, the focus is on the lifecycle steps for model development and operations as well as the management of models and their metadata. Although there are some functionality-related intersections with the class of experiment management systems, AMSs of this class often provide support for the Operations Stage.

As two of the first model-oriented AMSs, *ModelDB* [157, 154, 156, 159, 160] and *ModelHub* [99, 100, 101] both focus on supporting model development, deployment, and monitoring. While ModelDB versions models and their metadata in a relational database, ModelHub incorporates an ML artifact versioning system enriching and extending Git, and a read-optimized parameter archival storage that minimizes storage footprint using deltas and accelerates query workloads with minimal loss of accuracy.

ModelKB is an AMS with a focus on model management, experimentation, deployment, and monitoring [49, 51, 50]. It uses custom callbacks in native ML frameworks to collect metadata about each experiment and automatically generates source code for deployment, sharing, and reproducibility.

The *MMP* is a model management platform tailored to Industry 4.0 environments by associating ML models with business and domain metadata

[166]. It provides a model metadata extractor, a model registry, and a context manager to store model metadata in a central metadata store.

Compared to the previously discussed systems, *MISTIQUE* is specialized for the storage and management of model intermediates (e. g. input data, learned hidden representations) to accelerate model evaluation, performance diagnosis, and interpretability [155]. It decides for each diagnostic query whether to re-run the model or to read a previously stored intermediate and reduces storage footprint of model intermediates with storage optimizations such as quantization, summarization, and data de-duplication.

Motivated by fragmented ML workflows which require juggling between different programming paradigms and software systems, ML model training and inference algorithms as well as model management capabilities are increasingly integrated directly into DBMSs. *Sql4ml* enables expressing supervised ML models in SQL and translating them into Python code for training in TensorFlow [92, 93].

Vertica-ML is an ML extension on top of the distributed and parallelized RDBMS *Vertica Analytic Database* [44, 161, 82], which aims for eliminating the transfer of big volumes of data, avoiding the maintenance of a separate analytical system, and addressing concerns of data security and provenance by combining a full-fledged DBMS with the scalability and performance of in-database ML algorithms.

MLModelCI [176, 175, 30], *ModelCI-e* [62], *Clipper* [33, 34], *Rafiki* [164, 165], and *Overton* [124] are AMSs focusing on the Model Operations stage. MLModelCI is an MLOps platform for the automated deployment of pre-trained ML models and online model serving. Profiling under different settings (e. g. batch size and hardware) provides guidelines for balancing the trade-off between performance and cost. For the deployment to cloud environments, MLModelCI uses Docker. ModelCI-e is a plugin system for continual learning and deployment, enabling model updating and validation without model serving engine adaption. Clipper is a general-purpose low-latency model serving system. By exploiting caching, batching, and adaptive model selection techniques, Clipper reduces prediction latency and improves prediction throughput, accuracy, and robustness without modifying underlying ML frameworks. Rafiki provides a model training service supporting distributed hyperparameter tuning and a model inference service with online model ensembling that is amenable to the trade-off between latency and accuracy. Overton is an AMS focusing on building, deploying, and monitoring production models. It aims to support ML engineers in maintaining and improving model quality in the face of changes to the input distribution and new production features.

CMS is a container-based continuous learning and serving platform designed for industrial monitoring and analysis use cases [85]. Its primary goal is to simplify and automate the process of model generation, deployment, and switching. Building on top of the Kubernetes management platform Rancher, CMS provides resource-efficient orchestration of model training tasks and seamless model switching and serving without interruption of online operations and with minimal human interference.

ModelHub.AI is a community-driven platform for the dissemination of deep learning models [61, 106, 107]. It is founded on a container-based software engine that provides a standard template for models and exposes interfaces for model-specific functions as well as data pre- and post-processing. ModelHub.AI is domain-, data-, and framework-agnostic, catering to different workflows and contributor's preferences.

Dataset & Feature Management.

Complementary to the previously described class, this class focuses on support for the Data-oriented Stage by providing dataset, label, and feature storage and management capabilities as well as functionality and interfaces for data (pre)processing, feature selection and engineering, and provenance tasks.

MLdp is Apple's platform for managing ML data artifacts [3]. It provides a minimalist and flexible data model for integrating different varieties of data, a hybrid storage approach for large volumes of raw data and high concurrent updates on volatile data, version and dependency management, data provenance, and integration with major ML frameworks.

In comparison, *ExDra* provides an infrastructure for data acquisition, integration, and preprocessing from federated and heterogeneous raw data sources [19]. It uses SystemDS for federated linear algebra programs, parameter servers, and data processing pipelines. Trained models and their provenance are stored in a model management database.

Pachyderm is a data pipeline management platform [114, 115]. It provides automated data versioning, containerized pipeline execution, as well as immutable data lineage and provenance. Also motivated by enabling explainability, *ProvLake* is a data management system capable of capturing, integrating, and querying data across multiple distributed services, programs, databases, stores, and computational workflows by leveraging provenance data [141, 140, 139, 66, 117].

Unlike the previous systems, *Data Provenance Library* [27, 138] and *Shuffler* [151, 132] operate at the

library level. Data Provenance Library is a Python library for capturing and querying fine-grained provenance of data preprocessing pipelines. It is based on a formal model comprising data reduction, augmentation and transformation operators, as well as a MongoDB database as provenance store. Shuffler is a toolbox for data preparation workflows of computer vision tasks. It employs relational databases and SQL for storing and manipulating annotations.

Feast is a feature store for managing and serving ML features to models in production [47, 46]. Feast aims for enabling DevOps-like practices for the lifecycle of features. As a single source of truth, Feast serves feature data either from a low-latency online store for real-time prediction, or from an offline store for scale-out batch scoring or model training.

5.2 Discussion Along Criteria

This section discusses the assessment results comparatively along the criteria and subcriteria.

Lifecycle Integration.

Requirements engineering in the context of ML is a young field of research [5, 162, 163]. Many of the methods and approaches known from traditional software engineering have yet to be adapted for ML systems [16, 21, 91]. Additionally, the specification of non-trivial requirements often necessitates domain expert knowledge. Consequently, the tool support is still poor and requirements engineering functionality is not yet covered by the assessed AMSs.

In contrast, many of the systems and platforms studied do at least provide partial functional support or interfaces for the Data-oriented Stage: While most systems and platforms lack support for the Data Collection step, probably due to the individuality of data type, volume, sources, and collection approaches, many integrate functionality or provide functional interfaces for at least one of the Data Preparation & Cleaning, Data Labeling, and Feature Engineering & Selection steps (ca. 75 %).

Systems and platforms for lifecycle management offer the widest range of functions: Cloud platforms with integrated artifact management often provide their own tools with a graphical UI. Open-source systems such as ClearML or MLflow, on the other hand, offer interfaces for integrating user code, which can then be used within pipelines for automation. In particular, AMSs with a narrow focus stand out here: For example, Feast and Hopsworks provide feature stores that are designed specifically for ML feature selection, storage, processing, and distribution.

A large proportion of the systems and platforms assessed provide wide or complete support for the Model-oriented Stage (86 %), including both integrated functionality and interfaces to typical ML frameworks (TensorFlow, PyTorch, etc.), which provide functional support for model building, training, evaluation, and optimization. Nevertheless, some systems deviate from this due to their goals: for example, MMP and Vertica-ML do not support *Model Design* but have an integrated set of ML models.

The functional support for deployment and monitoring of models (*Operations Stage*) is quite heterogeneous: 27 of 64 (ca. 42 %) provide full support and 14 of 64 (ca. 22 %) partial support. A majority of the lifecycle management systems and platforms provide functionality for deploying models as web services (e. g. via REST interfaces) as well as continuous collection and monitoring of performance and quality metrics. In 6 out of 13 systems from the pipeline management class this is also the case, 3 further systems only provide support for deploying model serving environments. In the remaining 3 classes, the support is much lower due to the objective of the corresponding systems and platforms.

Artifact Support.

With more than 92 % on average, a large proportion of the systems and platforms takes model artifacts into account. The proportion for data-related artifacts is lower at ca. 80 % (support for at least one type). Model-specific metadata such as hyperparameters or metrics are collected and processed by more than 80 % of the systems. Experiment/project metadata and pipelines are supported by ca. 67 % respectively 55 % of all systems. Only every second system takes software artifacts into account. The kind of support is strongly dependent on the objective and system class and is very heterogeneous across all systems and platforms; the relationship to the supported operations must be considered.

Operations.

With more than 90 %, the majority of the systems and platforms offers artifact capturing and logging functionality. Depending on the use of repositories and comparable techniques, only less than a half enable snapshots and intermediate states of artifacts to be checked in and rolled out again. Also, over 90 % provide operations to query and retrieve stored artifacts. More than two-thirds (ca. 67 %) have comparison functionality, ca. 52 % provide artifact lineage, and only ca. 17 % offer provenance functionality. Visualization operations are present in ca. 62 % of the systems. While the presence of typical management operations such as modify, delete, and execute & run is quite common, deployment operations are only present in about half of all systems.

While the platforms and systems of the lifecycle, experiment, and model management classes mostly provide complete functionality for export and import as well as sharing with other collaborators, these functions are less prominent or not available at all for the other classes.

Collection & Storage.

The collection of artifacts can either require explicitly added instructions (subcriterion *Intrusive*), such as Python functions or callbacks, or be (semi-)automatic (subcriterion *Non-intrusive*). While exactly half of the systems provide both intrusive and non-intrusive collection of artifacts, primarily of the lifecycle, pipeline, and experiment management classes, almost two-thirds of all systems (ca. 64 %) at least support automatic collection.

The types of storage used are highly dependent on the goals and focus of a system or platform. For example, lifecycle management systems provide an appropriate type of storage for each type of artifact (see § 4 for related discussion). It is also recognizable for the other system classes that the supported storage types are related to the supported artifacts themselves. Exceptions to this are systems and platforms such as Velox or MMP, which are tailored to specific domains and for this reason only support limited number of dataset and model types, as well as MISTIQUE or sql4ml, in which deep learning models are storage based on individual data models or supervised ML models in relational table structures.

In total 27 of the 64 systems and platforms complementarity support both the complete versioning of a project or lifecycle state including any artifacts and the versioning of individual artifact snapshots. About half of all systems and platforms support at least repositories for versioning, whereby in addition to the general-purpose version control systems, variants tailored specifically to ML are increasingly being used, which, for example, perform effective model versioning using deltas and provide special commands for model, dataset, pipeline, and experiment comparison and lineage information, as demonstrated by ModelHub's dlv and DVC.

Interfaces & Integration.

Overall, many of the examined systems and platforms provide a wide range of interfaces and integration. While Python SDKs or REST APIs are provided by almost 9 out of 10 systems and platforms (ca. 88 %), CLI tools and web UIs are available in over two-thirds (ca. 66 % respectively ca. 73 %). Here, especially the lifecycle management and experiment management classes stand out.

The programming language support (ca. 94 %) – primarily Python as the data science quasi-standard –, the integration for ML and data science frameworks (ca. 92 %) as well as the direct or indirect support for interactive and collaborative notebooks (ca. 92 %) is pronounced across all classes of systems and platforms. In particular, because of their focus, the systems and platforms of the lifecycle, pipeline, and experiment management classes have the highest degree of functional integration, which is related to the first category Lifecycle Integration.

Operation & Licensing.

The capabilities for system and platform operation are highly dependent on the corresponding software architecture. 26 systems allow only one, 14 two, and 24 all three modes of operation. Over half of all systems can be used locally (ca. 54 %), either as a library, local server application, Docker container, or locally executable Kubernetes variant such as Minikube or Kind. Two-thirds of the systems (ca. 66 %) can be deployed on-premise (e. g. on a dedicated server or cluster) and three-quarters are capable of running in a cloud (ca. 75 %).

Among the systems and platforms studied, a total of 35 systems have some kind of free license, typically with source code freely available, and the remaining 32 have a non-free, proprietary license.

6. CONCLUSION

This paper discusses system support for ML artifact management as an essential building block to achieve comparability, reproducibility, and traceability of artifacts created and used within the ML lifecycle. Objectives, fields of application, and functional ranges are heterogeneous and the selection of AMSs is quite difficult. Based on a systematic literature review, we derive functional and non-functional criteria that enable the systematic assessment of AMSs. Using the criteria, we assess and discuss a comprehensive selection of 64 systems and platforms from academia and industry.

As complementary and future work, we aim to investigate system support for automating ML tasks, e. g. AutoML techniques such as automated hyperparameter optimization, neural architecture search, and meta-learning, as well as for establishing ML-related security properties, e. g. techniques for hardening against and preventing model exploratory, data poisoning, and evasion attacks.

7. ACKNOWLEDGMENTS

This work was partially funded by the Thuringian Ministry of Economic Affairs, Science and Digital Society (grant 5575/10-3).

8. REFERENCES

[1] Acusense Technologies, Inc. Datmo – Productivity for Data Science & AI, 2022. https://www.datmo.com.

[2] Acusense Technologies, Inc. datmo/datmo: Open source production model management tool for data scientists, 2022. https://github.com/datmo/datmo.

[3] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, V. Shah, B. Shen, L. Sugden, K. Zhao, and M. Wu. Data Platform for Machine Learning. In *SIGMOD '19*, pages 1803–1816, 2019.

[4] L. Aguilar, D. Dao, S. Gan, N. M. Gürel, N. Hollenstein, J. Jiang, B. Karlas, T. Lemmin, T. Li, Y. Li, X. Rao, J. Rausch, C. Renggli, L. Rimanic, M. Weber, S. Zhang, Z. Zhao, K. Schawinski, W. Wu, and C. Zhang. Ease.ML: A Lifecycle Management System for Machine Learning. In *CIDR '21*, 2021.

[5] K. Ahmad, M. Bano, M. Abdelrazek, C. Arora, and J. C. Grundy. What's up with Requirements Engineering for Artificial Intelligence Systems? In *RE '21*, pages 1–12, 2021.

[6] Allegro AI. allegroai/clearml: ClearML – Auto-Magical Suite of tools to streamline your ML workflow. Experiment Manager, ML-Ops and Data-Management, 2022. https://github.com/allegroai/clearml/.

[7] Allegro AI. ClearML – MLOps for Data Science Teams, 2022. https://clear.ml.

[8] Amazon. Amazon sagemaker, 2022. https://aws.amazon.com/de/sagemaker/.

[9] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software Engineering for Machine Learning: A Case Study. In *SEIP@ICSE '19*, pages 291–300, 2019.

[10] AMPLab. amplab/velox-modelserver, 2022. https://github.com/amplab/velox-modelserver.

[11] P. Andrews, A. Kalro, H. Mehanna, and A. Sidorov. Productionizing Machine Learning Pipelines at Scale. In *MLSys@ICML '16*, 2016.

[12] Apache. Zeppelin, 2022. https://zeppelin.apache.org.

[13] Apache Software Foundation. Apache Spark – Unified engine for large-scale data analytics, 2022. https://spark.apache.org.

[14] ArangoDB. ArangoML, 2022. https://www.arangodb.com/machine-learning/.

[15] ArangoDB. arangoml/arangopipe: ArangoML Pipeline is a common and extensible Metadata Layer for Machine Learning Pipelines based on ArangoDB, 2022. https://github.com/arangoml/arangopipe.

[16] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch. Software Engineering Challenges of Deep Learning. In *SEAA '18*, pages 50–59, 2018.

[17] AutoAI. autoai-org/AID: One-Stop System for Machine Learning, 2022. https://github.com/autoai-org/aid.

[18] A. Barrak, E. E. Eghan, and B. Adams. On the Co-evolution of ML Pipelines and Source Code – Empirical Study of DVC Projects. In *SANER '21*, pages 422–433, 2021.

[19] S. Baunsgaard, M. Boehm, A. Chaudhary, B. Derakhshan, S. Geißelsöder, P. M. Grulich, M. Hildebrand, K. Innerebner, V. Markl, C. Neubauer, S. Osterburg, O. Ovcharenko, S. Redyuk, T. Rieger, A. Rezaei Mahdiraji, S. B. Wrede, and S. Zeuch. ExDRa: Exploratory Data Science on Federated Raw Data. In *SIGMOD '21*, pages 2450–2463, 2021.

[20] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *KDD '17*, pages 1387–1395, 2017.

[21] H. Belani, M. Vukovic, and Z. Car. Requirements Engineering Challenges in Building AI-Based Complex Systems. In *REW '19*, pages 252–255, 2019.

[22] L. Biewald. Experiment Tracking with Weights and Biases, 2020. https://www.wandb.com.

[23] M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthör, K. Innerebner, F. Klezin, S. N. Lindstaedt, A. Phani, B. Rath, B. Reinwald, S. Siddiqui, and S. B. Wrede. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR '20*, 2020.

[24] S. Bozinovski. Reminder of the First Paper on Transfer Learning in Neural Networks, 1976. *Informatica*, 44:291–302, 2020.

[25] E. Brumbaugh, A. Kale, A. Luque, B. Nooraei, J. Park, K. Puttaswamy, K. Schiller, E. Shapiro, C. Shi, A. Siegel, N. Simha, M. Bhushan, M. Sbrocca, S. Yao, P. Yoon, V. Zanoyan, X. T. Zeng, Q. Zhu, A. Cheong, M. G. Du, J. Feng, N. Handel, A. Hoh, J. Hone, and B. Hunter. Bighead: A Framework-Agnostic, End-to-End Machine Learning Platform. In *DSAA '19*, pages 551–560, 2019.

[26] V. Chaoji, R. Rastogi, and G. Roy. Machine Learning in the Real World. *PVLDB*, 9(13):1597–1600, 2016.

[27] A. Chapman, P. Missier, G. Simonelli, and R. Torlone. Capturing and Querying Fine-grained Provenance of Preprocessing Pipelines in Data Science. *PVLDB*, 14(4):507–520, 2020.

[28] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *DEEM@SIGMOD '20*, 2020.

[29] K. Chen, H. Su, W. Chuang, H. Hsiao, W. Tan, Z. Tang, X. Liu, Y. Liang, W. Lo, W. Ji, B. Hsu, K. Hu, H. Jian, Q. Zhou, and C. Wang. Apache Submarine: A Unified Machine Learning Platform Made Simple. *CoRR*, abs/2108.09615, 2021.

[30] Cloud Application and Platform Lab, NTU. cap-ntu/ML-Model-CI: MLModelCI is a complete MLOps platform for managing, converting,

profiling, and deploying MLaaS (Machine Learning-as-a-Service), bridging the gap between current ML training and serving systems, 2022. https://github.com/cap-ntu/ML-Model-CI.
[31] Comet. Comet – Build better models faster, 2022. https://www.comet.ml.
[32] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR '15*, 2015.
[33] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI '17*, pages 613–627, 2017.
[34] D. Crankshaw, C. Zumar, J. Gonzalez, A. Tumanov, E. Sela, S. Mo, R. Durrani, and E. Sheng. ucbrise/clipper: A low-latency prediction-serving system, 2022. https://github.com/ucbrise/clipper.
[35] Data Science Platform. Data Science Workbench (CDSW) | Cloudera, 2022. https://www.cloudera.com/products/data-science-and-engineering/data-science-workbench.html.
[36] DataRobot, Inc. DataRobot Enterprise AI Platform, 2022. https://www.datarobot.com/platform/.
[37] Deepkit. Deepkit – the AI training suite, model debugger, and computation management, 2022. https://deepkit.ai.
[38] Deepkit. deepkit/deepkit-ml: The collaborative real-time open-source machine learning devtool and training suite – Experiment execution, tracking, and debugging. With server and project management tools, 2022. https://github.com/deepkit/deepkit-ml.
[39] Department of Software Engineering, University of Szeged, Hungary. sed-inf-u-szeged/DeepWaterFramework: A Python framework for machine learning model training, hyper-parameter search, configuration management, and result visualization, 2022. https://github.com/sed-inf-u-szeged/DeepWaterFramework.
[40] Determined AI. determined-ai/determined: Determined – Deep Learning Training Platform, 2022. https://github.com/determined-ai/determined.
[41] Determined AI. Distributed Deep Learning and Hyperparameter Tuning Platform – Determined AI, 2022. https://www.determined.ai.
[42] DS3 Lab. DS3Lab/easeml: A Scalable Auto-ML System, 2022. https://github.com/DS3Lab/easeml.
[43] Dunn, Jeffrey. Introducing FBLearner Flow: Facebook's AI backbone, 2016. https://engineering.fb.com/2016/05/09/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/.
[44] A. Fard, A. Le, G. Larionov, W. Dhillon, and C. Bear. Vertica-ML: Distributed Machine Learning in Vertica Database. In *SIGMOD '20*, pages 755–768. ACM, 2020.
[45] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Comm. of the ACM*, 39(11):27–34, 1996.
[46] Feast Authors. feast-dev/feast: Feature Store for Machine Learning, 2022. https://github.com/feast-dev/feast.
[47] Feast Authors. Feast: Feature Store for Machine Learning, 2022. https://feast.dev.
[48] R. Ferenc, T. Viszkok, T. Aladics, J. Jász, and P. Hegedüs. Deep-water framework: The Swiss army knife of humans working with machine learning models. *SoftwareX*, 12:1–7, 2020.
[49] G. Gharibi, V. Walunj, R. Alanazi, S. Rella, and Y. Lee. Automated Management of Deep Learning Experiments. In *DEEM@SIGMOD '19*, pages 8:1–8:4, 2019.
[50] G. Gharibi, V. Walunj, R. Nekadi, R. Marri, and Y. Lee. Automated end-to-end management of the modeling lifecycle in deep learning. *Empir. Softw. Eng.*, 26(2):17, 2021.
[51] G. Gharibi, V. Walunj, S. Rella, and Y. Lee. ModelKB: Towards Automated Management of the Modeling Lifecycle in Deep Learning. In *RAISE@ICSE '19*, pages 28–34, 2019.
[52] Google. google/ml-metadata: For recording and retrieving metadata associated with ML developer and data scientist workflows, 2022. https://github.com/google/ml-metadata.
[53] Google. Machine Learning Workflow, 2022. https://cloud.google.com/ai-platform/docs/ml-solutions-overview.
[54] Google. ML Metadata – TFX – TensorFlow, 2022. https://www.tensorflow.org/tfx/guide/mlmd.
[55] Google. TensorBoard, 2022. https://www.tensorflow.org/tensorboard.
[56] Google. TensorFlow, 2022. https://www.tensorflow.org.
[57] Google. Vertex AI, 2022. https://cloud.google.com/vertex-ai.
[58] K. Greff, A. Klein, M. Chovanec, F. Hutter, and J. Schmidhuber. The Sacred Infrastructure for Computational Research. In *SciPy '17*, pages 49–56, 2017.
[59] H2O.ai. H2O – H2O.ai, 2021. https://www.h2o.ai/products/h2o/.
[60] H2O.ai. h2oai/h2o-3: H2O is an Open Source, Distributed, Fast & Scalable Machine Learning Platform: Deep Learning, Gradient Boosting (GBM) & XGBoost, Random Forest, Generalized Linear Modeling (GLM with Elastic Net), K-Means, PCA, Generalized Additive Models (GAM), RuleFit, Support Vector Machine (SVM), Stacked Ensembles, Automatic Machine Learning (AutoML), etc., 2022. https://github.com/h2oai/h2o-3.
[61] A. Hosny, M. Schwier, C. Berger, E. P. Örnek, M. Turan, P. V. Tran, L. Weninger, F. Isensee, K. H. Maier-Hein, R. McKinley, M. T. Lu, U. Hoffmann, B. H. Menze, S. Bakas, A. Fedorov, and H. J. W. L. Aerts. ModelHub.AI: Dissemination Platform for Deep Learning Models. *CoRR*, abs/1911.13218, 2019.
[62] Y. Huang, H. Zhang, Y. Wen, P. Sun, and N. B. D. Ta. ModelCI-e: Enabling Continual Learning in Deep Learning Serving Systems. *CoRR*, abs/2106.03122, 2021.

[63] W. Hummer, V. Muthusamy, T. Rausch, P. Dube, K. E. Maghraoui, A. Murthi, and P. Oum. ModelOps: Cloud-Based Lifecycle Management for Reliable and Trusted AI. In *IC2E '19*, pages 113–120, 2019.
[64] IBM. Analytics Solutions Unified Method – Implementations with Agile principles. Rep., 2016.
[65] IBM. IBM Watson Machine Learning, 2022. https://www.ibm.com/de-de/cloud/machine-learning.
[66] IBM. ProvLake, 2022. https://ibm.biz/provlake.
[67] S. Idowu, D. Strüber, and T. Berger. Asset Management in Machine Learning: A Survey. In *SEIP@ICSE '21*, pages 51–60, 2021.
[68] IDSIA. IDSIA/sacred: Sacred is a tool to help you configure, organize, log and reproduce experiments developed at IDSIA, 2021. https://github.com/IDSIA/sacred.
[69] R. Isdahl and O. E. Gundersen. Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms. In *eScience '19*, pages 86–95, 2019.
[70] M. Ismail, E. Gebremeskel, T. Kakantousis, G. Berthou, and J. Dowling. Hopsworks: Improving User Experience and Development on Hadoop with Scalable, Strongly Consistent Metadata. In *ICDCS '17*, pages 2525–2528, 2017.
[71] Iterative. CML – Continuous Machine Learning, 2022. https://cml.dev.
[72] Iterative. CML Documentation, 2022. https://cml.dev/doc.
[73] Iterative. DVC: Data Version Control – Git for Data & Models, 2022. https://dvc.org.
[74] Iterative. DVC: Open-source Version Control System for Machine Learning Projects, 2022. https://dvc.org.
[75] Iterative. DVCLive Documentation, 2022. https://dvc.org/doc/dvclive.
[76] Iterative. Iterative – Developer tools for Machine Learning, 2022. https://iterative.ai.
[77] Iterative. Iterative Studio Documentation, 2022. https://dvc.org/doc/studio.
[78] Iterative. Iterative Studio: Git-based ML Experiments Management, 2022. https://studio.iterative.ai.
[79] Iterative. MLEM, 2022. https://mlem.ai.
[80] B. Karlas, J. Liu, W. Wu, and C. Zhang. Ease.ml in Action: Towards Multi-tenant Declarative Learning Services. *PVLDB*, 11(12):2054–2057, 2018.
[81] B. Kitchenham and S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE-2007-01, Keele University, 2007.
[82] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica Analytic Database: C-Store 7 Years Later. *PVLDB*, 5(12):1790–1801, 2012.
[83] LF Projects. MLflow – A platform for the machine learning lifecycle, 2022. https://mlflow.org.
[84] LF Projects. mlflow/mlflow: Open source platform for the machine learning lifecycle, 2022. https://github.com/mlflow/mlflow.
[85] K. Li and N. Gui. CMS: A Continuous Machine-Learning and Serving Platform for Industrial Big Data. *Future Internet*, 12(6), 2020.
[86] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang. Scaling Machine Learning as a Service. In *PAPIs '16*, pages 14–29, 2017.
[87] LinkedIn. Scaling Machine Learning Productivity at LinkedIn, 2019. https://engineering.linkedin.com/blog/2019/01/scaling-machine-learning-productivity-at-linkedin.
[88] LogicalClocks. Hopsworks: Enterprise Feature Store & End-to-End ML Pipeline, 2022. https://www.hopsworks.ai.
[89] LogicalClocks. logicalclocks/hopsworks: Hopsworks – Data-Intensive AI Platform with a Feature Store, 2022. https://github.com/logicalclocks/hopsworks.
[90] Z. Luo, S. H. Yeung, M. Zhang, K. Zheng, L. Zhu, G. Chen, F. Fan, Q. Lin, K. Y. Ngiam, and B. Chin Ooi. MLCask: Efficient Management of Component Evolution in Collaborative Data Analytics Pipelines. In *ICDE '21*, pages 1655–1666, 2021.
[91] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. In *XP '19*, volume 355 of *LNBIP*, pages 227–243, 2019.
[92] N. Makrynioti, R. Ley-Wild, and V. Vassalos. sql4ml: A declarative end-to-end workflow for machine learning. *CoRR*, abs/1907.12415, 2019.
[93] Makrynioti, Nantia. nantiamak/sql4ml: Translation of supervised machine learning models in SQL to TensorFlow code, 2022. https://github.com/nantiamak/sql4ml.
[94] N. Marz. How to beat the CAP theorem, 2011. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html.
[95] N. Marz and J. Warren. *Big Data*. Manning Publications, 2015.
[96] H. Miao. *Provenance Management for Collaborative Data Science Workflows*. Ph.d. diss., University of Maryland, 2018.
[97] H. Miao, A. Chavan, and A. Deshpande. ProvDB: Lifecycle Management of Collaborative Analysis Workflows. In *HILDA@SIGMOD '17*, pages 7:1–7:6, 2017.
[98] H. Miao and A. Deshpande. ProvDB: Provenance-enabled Lifecycle Management of Collaborative Data Analysis Workflows. *IEEE Data Eng. Bull.*, 41(4):26–38, 2018.
[99] H. Miao, A. Li, L. S. Davis, and A. Deshpande. ModelHub: Towards Unified Data and Lifecycle Management for Deep Learning. *CoRR*, abs/1611.06224, 2016.
[100] H. Miao, A. Li, L. S. Davis, and A. Deshpande. On Model Discovery For Hosted Data Science Projects. In *DEEM@SIGMOD '17*, pages 6:1–6:4, 2017.
[101] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards Unified Data and Lifecycle Management for Deep Learning. In *ICDE '17*, pages 571–582, 2017.
[102] Microsoft. Team Data Science Process Documentation, 2020. https://docs.microsoft.com/en-us/azure/

machine-learning/team-data-science-process/.
[103] Microsoft. Azure Machine Learning: Machine-Learning-as-a-Service, 2022. https://azure.microsoft.com/de-de/services/machine-learning/.
[104] MLCommons Association. mlcommons/mlcube: MLCube is a project that reduces friction for machine learning by ensuring that models are easily portable and reproducible, 2022. https://github.com/mlcommons/mlcube.
[105] MLCommons Association. MLCube, 2022. https://mlcommons.org/en/mlcube/.
[106] ModelHub.AI. Modelhub, 2022. http://modelhub.ai.
[107] ModelHub.AI. modelhub-ai/modelhub-engine: Backend library, framework, and API for models in modelhub), 2022. https://github.com/modelhub-ai/modelhub-engine.
[108] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, Y. Wu, Y. Zhu, and M. Weimer. Vamsa: Automated Provenance Tracking in Data Science Scripts. In *KDD '20*, pages 1542–1551, 2020.
[109] Neptune.ai. Neptune.ai – Metadata Store for MLOps, 2022. https://neptune.ai.
[110] Netflix, Inc. Metaflow, 2022. https://metaflow.org.
[111] A. Ng. NIPS 2016 Tutorial: Nuts and Bolts of Building Deep Learning Applications. In *NIPS '16*, 2016.
[112] A. A. Ormenisan, M. Ismail, S. Haridi, and J. Dowling. Implicit Provenance for Machine Learning Artifacts. In *MLSys '20*, 2020.
[113] A. A. Ormenisan, M. Meister, F. Buso, R. Andersson, S. Haridi, and J. Dowling. Time Travel and Provenance for Machine Learning Pipelines. In *OpML '20*, 2020.
[114] Pachyderm. Pachyderm – The Data Foundation for Machine Learning, 2022. https://www.pachyderm.com.
[115] Pachyderm. pachyderm/pachyderm: Reproducible Data Science at Scale!, 2022. https://github.com/pachyderm/pachyderm.
[116] A. Phani, B. Rath, and M. Boehm. LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. In *SIGMOD '21*, pages 1426–1439, 2021.
[117] Polyaxon. IBM/multi-data-lineage-capture-py: IBM Multi-Lineage Data System), 2022. https://github.com/IBM/multi-data-lineage-capture-py.
[118] Polyaxon. Polyaxon – machine learning at scale, 2022. https://polyaxon.com.
[119] Polyaxon. polyaxon/polyaxon: Machine Learning Platform for Kubernetes (MLOps tools for experimentation and automation), 2022. https://github.com/polyaxon/polyaxon.
[120] PostgreSQL Global Development Group. PostgreSQL: The world's most advanced open source database, 2022. https://www.postgresql.org.
[121] Project Jupyter. Project Jupyter, 2022. https://jupyter.org.
[122] S. Raschka. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. *CoRR*, abs/1811.12808, 2018.
[123] T. Rausch, W. Hummer, and V. Muthusamy. An Experimentation and Analytics Framework for Large-Scale AI Operations Platforms. In *OpML '20*, 2020.
[124] C. Ré, F. Niu, P. Gudipati, and C. Srisuwananukorn. Overton: A Data System for Monitoring and Improving Machine-Learned Products. In *CIDR '20*, 2020.
[125] C. Renggli, F. A. Hubis, B. Karlaš, K. Schawinski, W. Wu, and C. Zhang. Ease.Ml/Ci and Ease.Ml/Meter in Action: Towards Data Management for Statistical Generalization. *PVLDB*, 12(12):1962–1965, 2019.
[126] Replicate. Keepsake – Version control for machine learning, 2022. https://keepsake.ai.
[127] Replicate. replicate/keepsake: Version control for machine learning, 2022. https://github.com/replicate/keepsake.
[128] M. Salvaris, D. Dean, and W. H. Tok. *Deep Learning with Azure*. Apress, 2018.
[129] J. Schad, R. Sambasivan, and C. Woodward. Arangopipe, a tool for machine learning meta-data management. *Data Science*, 4(2):85–99, 2021.
[130] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. In *MLSys@NIPS '17*, 2017.
[131] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden Technical Debt in Machine Learning Systems. In *NIPS '15*, pages 2503–2511, 2015.
[132] Sentinent Technologies. kukuruza:shuffler: Studio – Toolbox for manipulating image annotations in computer vision, 2022. https://github.com/kukuruza/shuffler.
[133] Sentinent Technologies. StudioML – A Python model management framework, 2022. https://studio.ml.
[134] Sentinent Technologies. studioml/studio: Studio – Simplify and expedite model building process, 2022. https://github.com/studioml/studio.
[135] A. Sergeev and M. Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR*, abs/1802.05799, 2018.
[136] S. Shankar and A. Parameswaran. Towards Observability for Machine Learning Pipelines. *CoRR*, abs/2108.13557, 2021.
[137] C. Shearer. The crisp-dm model: The new blueprint for data minining. *Journal of Data Warehousing*, 5(4):13–22, 2000.
[138] Simonelli, Giulia. GiuliaSim/DataProvenance, 2022. https://github.com/GiuliaSim/DataProvenance.
[139] R. Souza, L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Vital Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. S. Netto. Workflow Provenance in the Lifecycle of Scientific Machine Learning. *Concurrency and Computation: Practice and Experience*, n/a(n/a):e6544, 2021.
[140] R. Souza, M. Mattoso, L. Azevedo, R. Thiago,

E. F. de Souza Soares, M. N. dos Santos, M. A. S. Netto, E. V. Brazil, R. Cerqueira, and P. Valduriez. Efficient Runtime Capture of Multiworkflow Data Using Provenance. In *eScience '19*, pages 359–368, 2019.

[141] R. Souza, P. Valduriez, M. Mattoso, R. Cerqueira, M. A. S. Netto, L. Azevedo, V. Lourenço, E. F. d. S. Soares, R. Thiago, R. Brandão, D. Civitarese, E. V. Brazil, and M. F. Moreno. Provenance Data in the Machine Learning Lifecycle in Computational Science and Engineering. In *WORKS@SC '19*, pages 1–10, 2019.

[142] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. S. Roselli, and N. Talagala. Model Governance: Reducing the Anarchy of Production ML. In *ATC '18*, pages 351–358, 2018.

[143] C. Sun, N. Azari, and C. Turakhia. Gallery: A Machine Learning Model Management System at Uber. In *EDBT '20*, pages 474–485, 2020.

[144] TensorHub, Inc. Guild AI – Experiment tracking, ML developer tools, 2022. https://guild.ai.

[145] TensorHub, Inc. guildai/guildai: Experiment tracking, ML developer tools, 2022. https://github.com/guildai/guildai.

[146] The Apache Software Foundation. Apache Submarine: Cloud Native Machine Learning Platform, 2022. https://submarine.apache.org.

[147] The Apache Software Foundation. Apache SystemDS – Declarative Large-Scale Machine Learning, 2022. https://systemds.apache.org.

[148] The Apache Software Foundation. apache/systemds: Apache SystemDS – A versatile system for the end-to-end data science lifecycle, 2022. https://github.com/apache/systemds/.

[149] The Kubeflow Authors. Kubeflow – The Machine Learning Toolkit for Kubernetes, 2022. https://www.kubeflow.org.

[150] The Kubeflow Authors. kubeflow/kubeflow: Machine Learning Toolkit for Kubernetes, 2022. https://github.com/kubeflow/kubeflow.

[151] E. Toropov, P. A. Buitrago, and J. M. F. Moura. Shuffler: A Large Scale Data Management Tool for Machine Learning in Computer Vision. In *PEARC '19*, 2019.

[152] J. Tsay, T. Mummert, N. Bobroff, A. Braz, P. Westerink, and M. Hirzel. Runway: machine learning model experiment management tool. In *SysML '18*, 2018.

[153] Valohai. Valohai MLOps Platform – Train, Evaluate, Deploy, Repeat, 2022. https://valohai.com.

[154] M. Vartak. MODELDB: A System for Machine Learning Model Management. In *CIDR '17*, 2017.

[155] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia. MISTIQUE: A System to Store and Query Model Intermediates for Model Diagnosis. In *SIGMOD '18*, pages 1285–1300, 2018.

[156] M. Vartak and S. Madden. MODELDB: Opportunities and Challenges in Managing Machine Learning Models. *IEEE Data Eng. Bull.*, 41(4):16–25, 2018.

[157] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. ModelDB: A System for Machine Learning Model Management. In *HILDA@SIGMOD '16*, 2016.

[158] VertaAI. loglabs/mltrace: Coarse-grained lineage and tracing for machine learning pipelines, 2022. https://github.com/loglabs/mltrace.

[159] VertaAI. MLOps Platform for Enterprise Data Science Teams – Verta, 2022. https://www.verta.ai.

[160] VertaAI. VertaAI/modeldb: Open Source ML Model Versioning, Metadata, and Experiment Management, 2022. https://github.com/VertaAI/modeldb.

[161] Vertica. In-database Machine Learning for Predictive Analytics at Scale, 2022. https://www.vertica.com/product/database-machine-learning/.

[162] H. Villamizar, T. Escovedo, and M. Kalinowski. Requirements Engineering for Machine Learning: A Systematic Mapping Study. In *SEAA '21*, pages 29–36, 2021.

[163] A. Vogelsang and M. Borg. Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In *REW '19*, pages 245–251, 2019.

[164] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad. Rafiki: Machine Learning as an Analytics Service System. *PVLDB*, 12(2):128–140, 2018.

[165] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad. nginyc/rafiki: Rafiki is a distributed system that supports training and deployment of machine learning models using AutoML, built with ease-of-use in mind, 2022. https://github.com/nginyc/rafiki.

[166] C. Weber and P. Reimann. MMP – A Platform to Manage Machine Learning Models in Industry 4.0 Environments. In *EDOCW '20*, pages 91–94, 2020.

[167] Weights & Biases. Weights & Biases – Developer Tools for ML, 2022. https://wandb.ai.

[168] T. Weißgerber and M. Granitzer. Mapping platforms into a new open science model for machine learning. *it Inf. Technol.*, 61(4):197–208, 2019.

[169] R. Wirth and J. Hipp. CRISP-DM: Towards a Standard Process Model for Data Mining. In *4th Int. Conf. on the Practical Applications of Knowledge Discovery and Data Mining*, pages 29–39, 2000.

[170] C. Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *EASE '14*, pages 38:1–38:10, 2014.

[171] X. Yao. MLPM: Machine Learning Package Manager. In *MLOps '20*, 2020.

[172] K. Yocum. kyocum/disdat: Data science tool for creating and deploying pipelines with versioned data, 2022. https://github.com/kyocum/disdat.

[173] K. Yocum, S. Rowan, J. Lunt, and T. M. Wong. Disdat: Bundle Data Management for Machine Learning Pipelines. In *OpML '19*, pages 35–37, 2019.

[174] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi,

S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[175] H. Zhang, Y. Huang, and Y. Li. Machine Learning Model CI, 2022. https://mlmodelci.com.

[176] H. Zhang, Y. Li, Y. Huang, Y. Wen, J. Yin, and K. Guan. MLModelCI: An Automatic Cloud Platform for Efficient MLaaS. In *MM '20*, pages 4453–4456, 2020.