# Asset Management in Machine Learning: A Survey

Samuel Idowu*, Daniel Strüber†, and Thorsten Berger*‡
*Chalmers | University of Gothenburg, Sweden
†Radboud University Nijmegen, Netherlands
‡Ruhr University Bochum, Germany

***Abstract*—Machine Learning (ML) techniques are becoming essential components of many software systems today, causing an increasing need to adapt traditional software engineering practices and tools to the development of ML-based software systems. This need is especially pronounced due to the challenges associated with the large-scale development and deployment of ML systems. Among the most commonly reported challenges during the development, production, and operation of ML-based systems are experiment management, dependency management, monitoring, and logging of ML assets. In recent years, we have seen several efforts to address these challenges as witnessed by an increasing number of tools for tracking and managing ML experiments and their assets. To facilitate research and practice on engineering intelligent systems, it is essential to understand the nature of the current tool support for managing ML assets. *What kind of support is provided? What asset types are tracked? What operations are offered to users for managing those assets?* We discuss and position ML asset management as an important discipline that provides methods and tools for ML assets as structures and the ML development activities as their operations. We present a feature-based survey of 17 tools with ML asset management support identified in a systematic search. We overview these tools' features for managing the different types of assets used for engineering ML-based systems and performing experiments. We found that most of the asset management support depends on traditional version control systems, while only a few tools support an asset granularity level that differentiates between important ML assets, such as datasets and models.**

***Index Terms*—machine learning, SE4AI, asset management**

## I. Introduction

An increasing number of software systems today implement AI capabilities by incorporating machine learning (ML) components. This growth increasingly demands using software-engineering methods and tools for systematically developing, deploying, and operating ML-based systems [1]–[3]. However, there are several difficulties associated with implementing these traditional methods in the ML application context. Arpteg et al. [2] characterize these fundamental issues of engineering AI systems into development, production, and organizational challenges. They highlight experiment management, dependency management, monitoring, and logging as part of the core issues under the development and production challenges. Moreover, engineers still face challenges to operationalize and standardize the ML development process [4], [5]. Hill et al.'s [4] field interview study reveals that support for tracking and managing the different ML assets is essential, with engineers currently resorting to custom or ad hoc solutions, given the lack of suitable management techniques. All their interviewees also revealed they are limited to versioning only the source code and not other ML assets, while many specified that they adopted informal methods, such as emails, spreadsheets, and notes to track their ML experiments' assets.

ML practitioners and data scientists are tasked with managing ML assets when iterating over stages of the ML process lifecycle (described in Section II-A). The development iterations, which are usually repeated until the process results in an acceptable model, increase the number of generated models and their associated assets. Reports [4], [6] show that ML practitioners usually generate hundreds of models before an acceptable model meets their target criteria. This iterative nature of developing ML systems contributes to the complexity of asset management in ML and calls for tool support to facilitate asset operations, such as tracking, versioning, and exploring. Specifically, we consider asset management in ML as a discipline that offers engineers the necessary management support for processes and operations on different types of ML assets. Various tools in the ML tools landscape, especially experiment management tools—the scope of our survey—offer asset management support. Specifically, an ML experiment can be described as a collection of multiple iterations over stages of an ML process lifecycle towards a specific objective. The experiment management tools aim to simplify and facilitate the model-building and management processes by tracking essential ML assets. The assets commonly used in a model building process include the dataset and source code used in producing the model, the source code used for feature extraction on the dataset, the hyperparameters used during model training, and the model evaluation dataset. ML experiment management tools usually offer APIs via which users can log assets of interest and their relationships for operations such as versioning, exploration, and visualization. Some experiment management tools offer support for drawing insights from experiment outcomes through visualization, while some offer execution-related supports such as reproduction of ML experiments, parallel execution, and multi-stage executions. Furthermore, note that ML practitioners and data scientists commonly use the term artifact to describe datasets and other model resources. In this work, we opt for the term *asset* to describe all artifacts used (or reused) during the ML process lifecycle.

To facilitate research and practice on engineering ML-based systems, it is essential to understand the support that the current ML tool landscape offers to engineers and data scientists for managing the diversity of ML assets. *What support do they provide? What are the typical asset types they track? What operations are offered to engineer on the assets? What are their commonalities and variabilities?*

In this paper, we discuss and position asset management as an essential discipline to scale the engineering of ML experiments and ML-based systems. We survey asset management support in 17 contemporary experiment-management tools, identifying the types of assets supported, and the operations offered to engineers for managing ML assets. Specifically, we conduct a feature-based survey, which is essentially a domain analysis to identify the characteristics of an application domain (defined by the subject experiment-management tools we survey). We model these characteristics as features in a feature model [7], [8], an intuitive tree-like notation commonly used in software variability management [9], [10]. In the literature, such feature-based surveys have been performed before to compare the design space of technologies, such as model transformations [11], conversational AI systems [12], language workbenches [13], or variation control systems [14]. Our study contributes a feature-model-based representation of tools with support for ML asset management—particularly the ML experiment management tools—that captures the asset types and the supported operations. We address the following research questions:

- **RQ1**: What asset types are tracked and managed in the subject tools?
- **RQ2**: What are the asset collection mechanisms used in collecting these assets?
- **RQ3**: How are the assets stored and version-controlled?
- **RQ4**: What are the management operations offered by the subject tools?
- **RQ5**: What integration support do they offer to other ML development systems?

Our study comprised collecting and selecting relevant tools for this study. We focused on experiment management tools, which typically offer management support for different ML asset types across stages of the ML process lifecycle. Specialized management tools, such as model-specific management tools (e.g., model registries and model databases [15]), dataset-specific management tools, pipeline or run orchestration-specific management tools, hyper-parameter management tools, visualization-specific tools, and experiment metadata databases [16] were beyond the scope of our study.

We hope that our study contributes to an increased empirical understanding of the solution space of ML asset management. ML practitioners and data scientists can use our survey results to understand the asset management features provided by contemporary experiment management tools. Researchers can identify gaps in the tool support for ML asset management, as well as they can classify their new techniques against our taxonomy (the feature model). Lastly, we hope that our result will contribute towards building tools with improved ML management support that promote traditional software engineering methods in developing ML-based systems.

## II. Asset Management

We now describe ML asset management as an essential discipline and position it by discussing background on ML process lifecycles and ML assets.

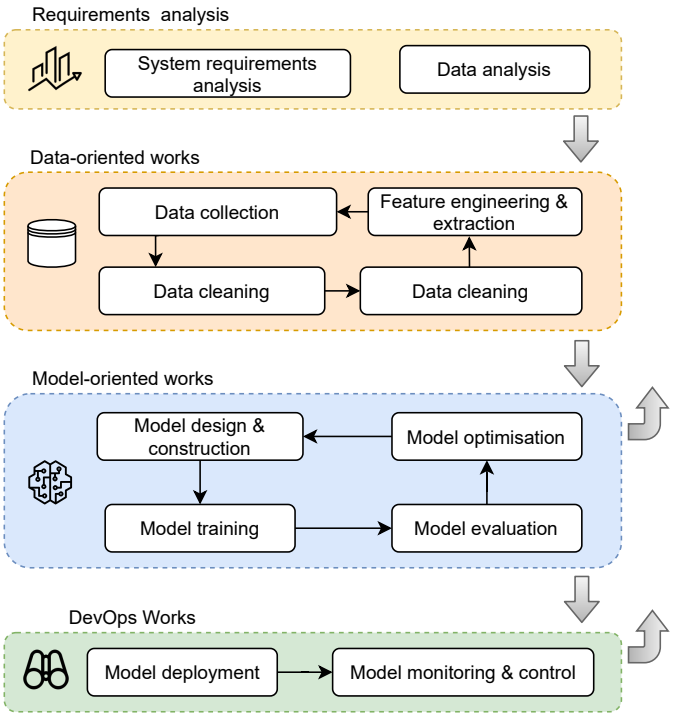


Fig. 1. Stages in a typical ML workflow

### A. ML Process Lifecycle

The traditional software engineering process [17] includes activities such as requirements analysis, planning, architecture design, coding, testing, deployment, and maintenance. Similarly, ML follows a set of well-defined processes that are grounded in workflows designed in the data science and data mining context. Examples include CRISP-DM [18], KDD [19], and TDSP [20]. Figure 1 shows a simplified workflow diagram of a supervised ML process lifecycle, structured along groups of development stages. The workflow consists of stages for requirements analysis, data-oriented works, model-oriented works, and DevOps works [1]. The requirements analysis stages involve analyzing the system requirements and 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 the deployment of ML models, monitoring and controlling of in-production models. Figure 1 illustrates multiple feedback loops (indicated by the upward arrows) from the model-oriented and DevOps stages to earlier stages. The feedback loop demonstrates iteration over sets of ML stages for a variable number of times until the process results in a model that meets a target objective. A need for asset management support is often attributed to the complexity and time overhead that arises with manually managing the large number of assets resulting from this iterative process [4], [6], [21].

### B. ML Artifacts & Assets

The term asset is conventionally used for an item that has been designed for use in multiple contexts [22], such as a

design, a specification, source code, a piece of documentation, or a test suite. Consequently, we use the term asset for an artifact that, after its initial use, is retained for future use. ML practitioners and data scientists often use the term artifact to describe required resources during an ML model development. These artifacts all qualify as assets in ML engineering because of the experimental nature and feedback loops in typical ML workflows, which requires keeping artifacts for future use. Conventional software engineering often has fewer asset types to manage than the more extensive diversity of assets under ML engineering. Conventional software engineering mostly deals with textual artifacts, while ML includes additional artifact types, such as datasets, learned models, hyper-parameters, and model performance metrics .

### C. Asset Management

In this light, we define asset management as an essential discipline for scaling the engineering of ML-based systems and experiments:

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

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 comprises activities pertaining to the practice areas dataset management, model management, hyper-parameter management, process execution management, and report management.

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

*b) Model development management:* Management tools in this area focus on model-oriented works of the ML lifecycle. They provide several supervised and unsupervised learning methods, such as classification, regression, and clustering algorithms to generate and evaluate ML models. The ML community has 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).

*c) Model storage and serving management:* Tools under this area focus on model-operation works of the ML process lifecycle. 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 performance of models (e.g., ModelDB).

*d) Hyper-parameter optimization management:* Searching or tuning for optimal hyper-parameters for a given ML task can be tedious. Tools in this area (e.g., Optuna) manage ML learning parameters and provide systematic ways to quicken the process of finding well-performing hyperparameters.

*e) Pipeline & run orchestration management:* Tools in this area (e.g., AirFlow, Luigi, Argo) provides functionalities to orchestrate the automatic execution of ML lifecycle stages as described in Fig. 1, from data collection to model serving. They often allow users to specify workflows as *Direct Acyclic Graphs* (DAGs) to form collections of ML stages represented in a way that describes their dependencies with other ML assets. Also, they often adopt containerized technologies to support distributed and scalable ML operations. Training, testing, deploying, and serving models are examples of ML operations that benefit from using run orchestrators for faster model training and inference.

*f) Reports & visualizations:* In this area, tools (e.g., TensorBoard, OmniBoard) present assets such as model evaluation metrics in graphical web dashboards to provide insight into ML experiments outcomes.

## III. Survey Methodology

We now describe the methodology behind our survey of asset management capabilities within experiment management tools used in practice. Specifically, we systematically selected our candidate tools, and analyzed them to arrive at a feature model to answer our research questions.

### A. Tool Selection Process

Following established guidelines for systematic reviews [23], we assessed existing ML experiment management tools for their capabilities. Selecting the tools was a four-step process:

- First, we defined our search topic based on our research questions and accordingly chose our search terms.
- Second, we designed our search strategy and carried out our search on data sources (described shortly).
- Third, we identified and applied the initial selection criteria $C_1$ to collate a preliminary list of all identified ML management tools.
- Last, we identified and applied the additional selection criteria $C_2$ to arrive at the list of ML experiment management tools as the final candidate tools considered in this study.

Figure 2 presents a summary of the selection process, and the following subsections provide further details on our selection process.

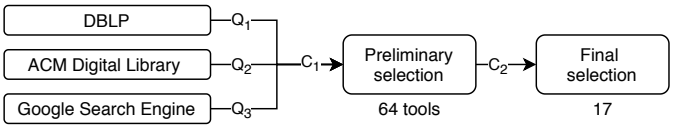


Fig. 2. Overview of our selection process.

*1) Data Source and Search Strategy:* We employed three data sources in our work, namely DBLP, ACM digital library, and Google search engine. Our search was conducted within the strategy guidelines presented by Kitchenham and Charters [23]. Accordingly, we excluded search terms such as "asset" from our search queries because we got several non-related results and instead used terms derived from our research questions and known related literature [6], [24]–[28]. We also guided our search with our knowledge and experience of ML, and its application [12], [29]–[34]. The following describes our search for each data source.

***Source***$_1$ (DBLP): First, we searched using DBLP,—a comprehensive and high-quality bibliographic data source that only allows title search. From the research questions' related terms, we created a search query, $Q_1$ ("machine learning" & ("reproducible" | "reproducibility" | "reusable" | "provenance" | "lifecycle")), which resulted in a total of 46 publications.

***Source***$_2$ (ACM digital library): Since DBLP allows search on only the literature title, we searched both literature titles and abstracts using ACM digital library. Similar to the approach used for $Source_1$, we created a search query, $Q_2$ ("machine learning" AND ("reusable" OR "lifecycle" OR "reproducibility" OR "provenance")), based on relevant terms. Our search produced 12 and 127 results from the literature title and abstract search, respectively.

***Source***$_3$ (Google search engine): Since we expected that many ML asset management tools used in practice are not documented in research publications, we performed an internet search using the search query, $Q_3$ ("machine learning") AND ("artifacts" OR "experiments") AND ("provenance" OR "versioning" OR "tracking" OR "history") AND ("Reusable" OR "reproducible") AND "management" AND ("framework" OR "tool" OR "platform"). From our search, we obtained 181 results. Note that there is a well-known phenomenon where Google search reports a significantly larger number of results than the actual result count. In our case, Google search initially reported over 2 million results, which later decreased to 181 results when we navigated to the last result page.

Using the literature we found from $Source_1$ and $Source_2$, we performed backward snowballing until we found no new relevant tools from the last paper. We manually filtered based on our selection criteria $C_1$ (described shortly) on results from all sources ($Source_1$, $Source_2$ and $Source_3$) when collating asset management tools from the data sources. After that, we obtained a preliminary list of 66 ML asset management tools discarding duplicate entries. Finally, we selected using criteria $C_2$ (described shortly) to arrive at our final selection of tools with ML asset management support.

*2) Selection Criteria:* As proposed by Kitchenham and Charters [23], we describe the inclusion and exclusion criteria used in this survey to filter out and define the scope of tools that we analyzed. Our selection criteria ensured that we consider all relevant assets management tools to discover findings that pertain to our research questions. Since we filtered at two different stages of our selection process, we tagged the selection criterion $C_1$ and $C_2$. The prior indicates criteria applied when collating all tools with asset management support found from our data sources, while the latter indicates those applied to the preliminary selection to derive the experiment management tools (see Fig. 2).

TABLE I
LIST OF SELECTED TOOLS WITH ML ASSET MANAGEMENT SUPPORT.

| 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) | - |

**Inclusion criteria:** We considered the following:

- Tools that covers any of the ML asset management areas described in Section II-C. ($C_1$)
- Tools with meaningful prominence measured by search trends and/or GitHub stars ratings. ($C_1$)
- Tools with the primary purpose of ML experiment management (i.e., tools specifically designed to track and manage ML experiments and their assets) as we recognize them to be the most comprehensive with coverage of all asset management areas, and can provide insight into the ML asset management domain space when empirically examined. ($C_2$)

**Exclusion criteria:** We excluded the following:

- Proposed frameworks or prototypes from literature. ($C_1$)
- Tools that lack well-defined documentation in English language. ($C_1$)
- General ML frameworks or ML development tools such as Scikit [35] and TensorFlow [36]. ($C_1$)
- Specialized tools for a single management area, such as dataset, ML model, hyper-parameter, pipeline, or execution orchestration management. ($C_2$)

Table I shows our final 17 tools we evaluated in this work.

### B. Analysis of Identified Tools

This study aims to recognize the characteristics that differentiate our subject tools using features [37] and to represent them in a feature model [7], [8]. This analysis process has been divided into different stages to study our candidate tools' management capabilities and build a resulting feature model. Our analysis is based on information found in publicly available documentation for each of the tools, and in a few cases, we had to test the tools for their available functionalities when needed practically.

First, we have performed an initial analysis of a single ML tool to identify its supported ML asset types, the collection approaches, the storage options, supported asset operations, and integration capabilities. We partly established the terminologies

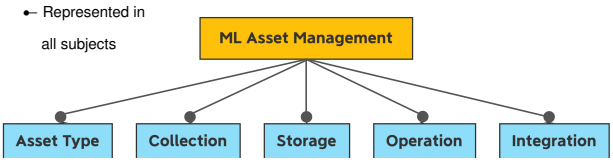


Fig. 3. Main ML asset management features.

to be used in our models. This stage produced our baseline version of the feature model that we present as our contribution.

Second, building on the first draft version, we adopted an iterative process to evaluate additional tools while modifying terminologies and the model structure to accommodate variations from the new tools being assessed.

Lastly, at the end of the final iteration, all the authors met to review the latest structure for direct feedback on terminologies used and the feature model. We integrated this feedback into our survey to arrive at the contribution of this study.

## IV. Asset Management Features

We propose a feature model—outlined in Figs. 3-8—to characterize and describe the asset management support of our subjects. The top-level features—`Asset Type`, `Collection`, `Storage`, `Operation` and `Integration` —capture the core functionalities of the subjects in our study. `Asset type` outlines the data types that are tracked by our subjects; `Collection` describes how the assets are collected; `Storage` describes how the assets are stored and versioned; `Operation` specifies what operation types are supported; and `Integration` shows the subjects' integration support to other systems.

We describe these top-level features and their corresponding sub-features in the following subsections.

### A. Asset Type (RQ1)

Following the definition of the term asset, as presented in Section II, we define the feature `Assets Type` as the set of data types tracked and managed by our subjects. Contrary to traditional software engineering, whose primary asset type is source code, ML engineering has more diversified asset types, such as datasets, hyper-parameters used in training, trained models, and evaluation metrics. As shown in Fig. 4, `Resources`, `Software`, `Metadata`, and `ExecutionData` are sub-features of `Asset type`.

*1) Resources:* Resources, also commonly referred to as 'artifacts' by many of the subjects, are the asset types required as input or produced as output from an ML workflow's stage (see Fig. 1). The subjects track resources with varying abstraction levels from specific asset types to `Generic` ones. We identified `Dataset` and `Model` as the most critical resource types. Many of them allow users to log the location and hash of data stored on local filesystems or cloud storage systems, such as AWS S3, Azure Storage, and Google Cloud Storage.

*a) Dataset:* The feature `Dataset` is available for subjects that identify datasets as an asset type. Data is an essential asset type in machine learning. Most of the ML workflow's stages, such as data collection, data transformation, feature extraction, model training, and evaluation, are data-dependent. The version of datasets used in ML workflow stages can be tracked to provide data lineage information for ML experiments.

*b) Model:* The feature `Model` is available for subjects that identify models as an asset type. ML models are created by learning from datasets using learning algorithms provided by ML development frameworks. Models are tracked along with their associated assets to facilitate result analysis, such as comparing models from different experiment runs.

*c) Environment:* Tatman et al. [38] reveal that sharing an environment with source code and dataset provides the highest level of reproducibility. With feature `Environment`, users can track environment resources such as Docker containers or Conda environments as experiments' assets to ensure reproducible ML experiments.

*d) Generic:* Several subjects lack dedicated support for tracking `Dataset` and `Model` types; instead, they provide a "one-size-fits-all" tracking of generic resources. Consequently, subjects with feature `Generic` can track all asset types that are required or generated during an ML experiment without differentiating them.

*2) Software:* This represents the software implementation of the ML process. `Software` typically involves the implementation of one or more stages of an ML workflow, and heavily relies on the supporting ML frameworks or model development tools that provide a collection of general ML techniques such as SciKit-Learn, PyTorch, TensorFlow, and Keras . We identified the sub-features of `Software` as `Notebook`, `SourceCode`, and `Parameter`.

*a) Notebook:* Similar to source code, notebooks contain the implementation to carry out specific ML operations. `Notebooks`, written in multiple execution cells, are usually used for small-scale, exploratory, and experimental ML tasks, where it is difficult to achieve acceptable software engineering practices such as modular design or code reuse. Notebooks (e.g., Jupyter [39]) are crucial for reproducible machine learning

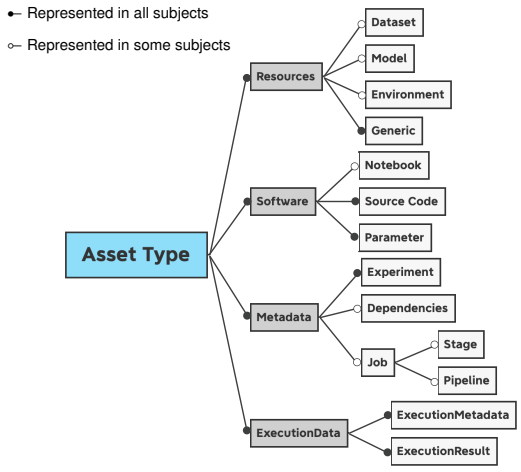


Fig. 4. Asset type feature model: A representation of the data types tracked by the subjects under study.

workflows that require literate and interactive programming. The `Notebook` feature indicates the support to track notebooks as an asset type. Users can track or version notebooks using snapshots or through Notebook checkpoints.

*b) Source Code:* This feature represents text-based files with implementation to carry out specific ML operations. Managing source code (or scripts) is generally less challenging than notebook formats for functional and large-scale engineering of ML-based systems because of available IDEs to support code assistance, dependency management and debugging [40]. Besides, source code are text-based files; therefore, they are easily version-controlled using traditional repositories. Consequently, ML practitioners and data scientists working on large scale systems often employ notebooks for initial ML experimentation and later convert them to source code files.

*c) Parameter:* Hyper-parameters are parameters utilized to control the learning process of an ML algorithm during the training phase of a model from a dataset (e.g., learning rate, regularization, and tree depth). Hyper-parameters are commonly tracked to facilitate the analysis of ML experiments' results. Some subjects (e.g., Comet.ml, Polyaxon, and Valoh.ai) provide hyper-parameter tuning and search features to facilitate the model-oriented stages of an ML workflow. In addition to hyper-parameters, the `Parameter` asset type also represents other configurable parameters that may influence an ML process.

*3) Metadata:* Metadata is a vital part of information management systems since it allows the semantic description of entities. In our context, as a sub-feature of `Asset Type`, `Metadata` represents the descriptive and structural static information about ML experiments, their dependencies, and how they are executed.

*a) Experiment:* The `Experiment` represents the main asset-type to which other assets are associated. It is the core abstraction of experiment management tools. Other tracked assets are usually linked with an `experiment`.

*b) Dependencies:* These are metadata information about the environment dependencies of an experiment. Examples include environment variables, host OS information; hardware details; Python libraries, and their versions.

*c) Jobs:* This feature represents the execution instructions of an ML experiment and how 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 their `Jobs`. Following from the ML workflow, the feature `Job` can be described as a `Stage` or a `Pipeline`. Listing 1 shows the representation of a stage and pipeline in DVC. Using CLI command as execution instruction by some subjects can also be seen as a different form of `Job` representation.

- A `Stage` is a basic reusable phase of ML workflow as illustrated in Fig. 1. They are defined with pointers to its required assets, such as `SourceCode`, `Parameters`, and input `Resources` (e.g., datasets).
- A `Pipeline` represents a reusable relationship between multiple stages to produce an ML workflow variants described in Fig. 1. ML pipelines are usually built as dependency graphs, where it uses input and output resources as dependencies between stages. In Listing 1, a dependency graph is represented with *featurize* stage which depends on the output of the *prepare* stage.

```
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 of `Metadata` representation of a `Pipeline` with two `Stages` in our subject DVC.

*4) ExecutionData:* This feature represents execution-related data that are tracked explicitly or automatically during the execution of an ML experiment. We identified `ExecutionMetadata` and `ExecutionResult` as sub-feature of `ExecutionData`.

*a) ExecutionMetadata:* This feature represents information about the execution process that is usually captured while execution (e.g., model training) is ongoing. These include terminal outputs, execution duration, events, statuses, and hardware consumption, such as CPU, GPU, and memory utilization.

*b) ExecutionResult:* This feature represents the assets generated as the results of an experiment. For a model training stage, this usually refers to evaluation metrics and can be tracked in different forms based on the ML task (e.g., sensitivity or ROC values for classification tasks; MSE, MAPE, or $R^2$ for regression tasks). For model training and data-oriented stages, `Model` and `Dataset` assets types are the results; hence, this indicates a relationship between the feature `ExecutionResult` and feature `Resources`.

**What asset types are tracked and managed? (RQ1)**

*Our subjects support resources (including datasets, models, environments, and generic resources); software (including notebooks, source code, and parameters); metadata (including experiment, dependencies, stages, and pipelines); and execution data (including execution metadata and results).*

### B. Collection (RQ2)

Feature `Collection`, shown in Fig. 5, represents the options provided by our subjects to track ML assets. The feature `Intrusiveness` shows the level of the explicit declaration

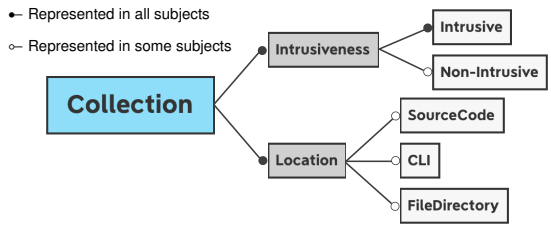


Fig. 5. Collection feature model: A representation of collection features used in tracking the asset types described in Section IV-C.

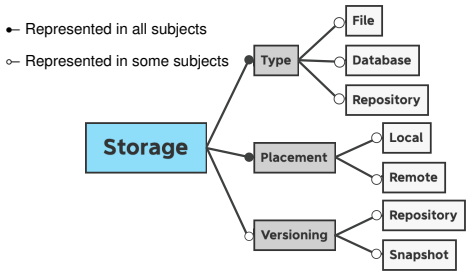


Fig. 6. Storage feature model: A representation of the storage feature observed from the subjects under study.

required to collect the assets; while feature `Location` represents the point where assets are being collected.

*a) Intrusiveness:* This describes the amount of instrumentation required by users to track assets. The `Intrusive` collection is invasive and requires users to add specific instructions and API calls in source code to track or log desired assets. In contrast, the `Non-Intrusive` collection automatically tracks or logs assets without the need for explicit instructions or API calls.

*b) Location:* This describes the collection point of assets. Assets can be extracted from `SourceCode`, `CLI` or `FileDirectory`. The common collection point across the considered subjects is the `SourceCode`, where the subjects provide a library and API that can be invoked to log desired assets within source code implementation. For collection at the `CLI`, subjects that are invoked via CLI commands allow users to specify pointers to assets as command arguments. In some subjects, CLI output are parsed to obtain the `ExecutionData's` assets. With the `FileDirectory` approach, subjects monitor assets from structured or instrumented file systems. Assets collected through this method are usually `Non-Intrusive` as modifications are automatically tracked.

**How are assets collected? (RQ2)**

*The collection approach can be intrusive or non-intrusive. Assets are commonly collected from source code; other collection locations include CLI arguments or logs, and instrumented file systems.*

### C. Storage (RQ3)

The feature `Storage` describes how the assets are stored and version controlled, and Fig. 6 shows its sub-features.

*a) Storage Type:* The feature `Storage` is fundamental to the observed subjects, especially the cloud services, which also provides cloud storage capabilities for generated assets. We identified `File`, `Database`, and `Repository`-based type of storage.

*b) Storage Placement:* The primary option provided for storage placement are `Local` and `Remote` placement with respect to the management tools. Assets stored remotely are usually tracked through identifier pointers and are 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.

*c) Versioning:* Support for `Versioning` is required to track the evolution of assets by keeping versioned assets during ML development. This feature is supported either by delegating asset versioning to traditional version control `Repository`, or via `Snapshot` of assets at each checkpoint. For the `Snapshot` approach, subjects independently create and track versions of assets, such as `Notebooks` and sub-features of `Resources`. In contrast, the feature `Repository` represents the collection and tracking of repository information (e.g., the commit hash and commit messages) with the associated experiment.

**How are assets collected? (RQ3)**

*The assets are either stored in file systems, databases, or repositories, either locally or remotely, while assets are version-controlled internally through snapshots or delegated to existing traditional repositories, such as Git.*

### D. Operations (RQ4)

We identified several operations supported by our subjects and represent them by the feature `Operation`. The `Execute` and `Explore` operations are the primary features supported by all subjects. Fig. 7 shows the sub-features of `Operation`.

*a) Explore:* The feature `Explore` represents the presence of operations that help derive insight and analyze experiment results. The subjects support various ways to `Query` assets, from listing all experiment assets to selection based on model performance filters. `Diff` indicates the presence of diffing between two or more assets while `Visualize` indicates the use of graphical presentations (e.g., charts) of experiments and their associated assets.

*b) Version:* This feature represents versioning-related operations. The feature `Commit` represents the presence of operations to create experiment checkpoints; while `Restore` indicates an operation to revert to an earlier version.

*c) Manage:* The feature `Dependency` represents the presence of dependency management, which is often supported by tools with `Stage` and `Pipeline` features as multi-stage execution. As an example, the *deps* in Listing 1 indicate the assets that are required for each stage. The `Execute` feature indicates the management of how ML experiments are being

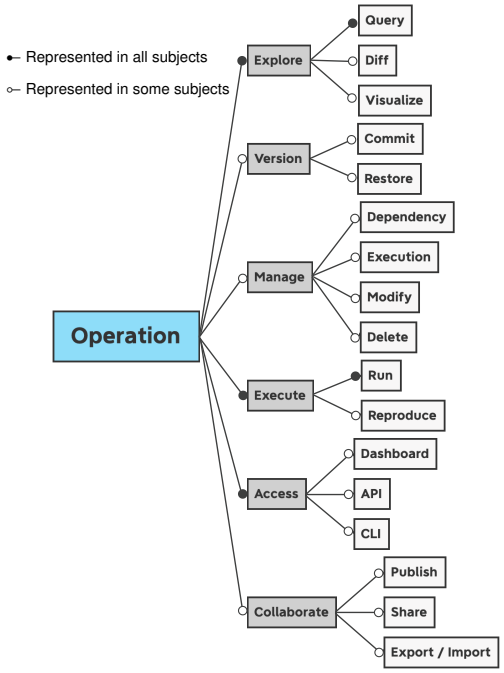


Fig. 7. Operation feature model: A representation of operations offered by the subjects under study.

executed, similar to run-orchestration-specific tools. Other sub-features of the feature `Manage` include `Modify` which provides the option to revise already logged assets (mostly the metadata of experiments), while `Delete` offers the option to remove already stored assets of an experiment.

*d) Execute:* The feature execute represents operations that invokes ML experiment via a defined entry-point. The features `Run` and `Reproduce` allow the execution of new and the reproduction of prior experiments, respectively.

*e) Access:* The mostly supported approach to `Access` stored assets is via graphical `Dashboards`. Other means of access include `API`, which provides REST interfaces or programming language APIs to access assets stored by the subject; the feature `CLI` exists for subjects that provide CLI commands for asset management.

*f) Collaborate:* This feature represents the presence of collaboration features, which are targeted at teams that need to share assets and results among the team members. User can `Publish`, `Share`, `Export` or `Import` experiment results or other required assets.

**What are the supported operations? (RQ4)**

*The supported operations allow users to explore assets using queries, comparison and, visualization of assets for insights. Other operations allow users to version assets, manage dependencies of assets and how they are executed. Users can also modify assets; run or reproduce experiment, access stored assets, and collaborate by publishing, sharing, exporting, or importing of assets.*

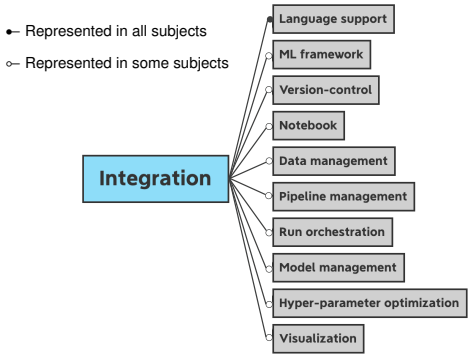


Fig. 8. Integration feature model: A representation of the integration support offered by the subjects under study.

### E. Integration (RQ5)

The experiment management tools are usually a piece of a larger toolchain. In large-scale ML production, several ML tools and frameworks are employed to develop, deploy, operate, and monitor ML models. The `Integration` feature, illustrated in Fig. 8, identifies the integration support type commonly provided by our subjects.

The feature `Language support` indicates support for programming languages, most notably Python. Many of our subjects support asset `collection` from source code; hence, they provide support for common `ML Frameworks` such as TensorFlow and SciKit-Learn. Such support allows some of our subjects to offer non-intrusive asset collection. Subjects that delegate to or depend on traditional versioning systems to version assets provide `Version-control` integration support. Certain subjects also provide integration for `Notebooks` in the form of plugins. `Data Management` indicates the integration support to retrieve and store assets from and to data specific tools and frameworks. The integration supports for workflow and execution orchestration systems are indicated by `Pipeline Management` and `Run Orchestration`; while the integration support for model-specific management tools is indicated by feature `Model Management`. `Hyper-parameter Optimization` indicates the presence of support for parameter search and tunning tools; while `Visualization` indicates the presence of integration for visualization libraries, such as Omniboard, and TensorBoard.

**What integration support is offered? (RQ5)**

*The subjects provide APIs, most frequently, Python-based ones. Further support includes ML frameworks, versioning systems, notebooks, data and model management tools, pipeline management and run orchestration tools, hyper-parameter optimization tools, and visualization libraries.*

## V. Discussion

**Version Control Systems.** Most of the subjects in this study delegate the versioning of ML assets to version control systems, such as Git. This approach increases tooling complexity for

users and can be a deterring factor for adoption. As shown in existing studies [4], [6], a large percentage of data scientists and ML practitioners still use solutions best described as ad hoc, while failing to achieve systematic ways of managing ML assets. A homogeneous way of ML asset management, where a user can manage 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 version control systems to a system that can support more ML asset types beyond text-based ones. Besides, the willingness to adopt these tools can be affected by the development context. For example, practitioners working on large ML projects that require collaboration and often generate hundreds of models are more likely to utilize asset management systems.

**Implicit Collection.** We observe that most subjects' asset collection methods are intrusive, i.e., it requires the users to instrument their source code to track asset information. This method is tedious and error-prone and can also be a deterring factor in adopting tools with management support. The non-intrusive asset collection methods—supported by subjects such as MLFlow and Weights & Biases for specific ML general frameworks, such as TensorFlow and SciKit Learn—eliminate these drawbacks. Ormenisan et al. [25] also proposed an implicit method of asset collection based on an instrumented file system and promises to solve issues associated with the intrusive asset collection methods.

**Reusability.** Reproducibility is one of the main objectives of using ML experiment management tools, and there is a significant presence of such features across most subjects in this study. In contrast, the reusability, which can significantly impact model development, is not often addressed. Few of the subjects provide a limited operation to reuse assets under the `Manage::Execution` operation, where users can define how experiment jobs should be executed. We observe that only the asset type `Job::Stage` and `Job::Pipeline` are provided with such operation, where the execution path of an ML pipeline skips unmodified stages when reproducing a pipeline. Reusability of more ML assets can significantly reduce model development time, enhance asynchronous collaboration in development teams, and motivate ML model evolution use-cases. We expect to see more tools addressing the reusability challenge of ML assets in the future.

## VI. Threats to Validity

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

**Internal Validity.** We manually selected the considered tools. One threat to the internal validity might be that the collection and filtering are subjective to individual opinion. We minimize this threat by validating our selection with information from the grey literature, such as market analysis reports. Since we consider a rapidly evolving technology landscape, we provide the snapshot date of accessed information. Furthermore, our internet exploration using the Google search engine is prone to varying results based on user, time, and search location—personalized user experience. This issue threatens the ability to reproduce the same search by other researchers. To mitigate these threats, we relied on multiple data sources to increase the reliability of our data collection process. For the cloud-based services considered in our work, we are limited to available online information. Consequently, we are unable to 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. [41] applies to our study.

## VII. Related Work

There are currently a few numbers of existing surveys and comparisons of tools with asset management support. We expect more studies in the future as discussions on standardized ML asset management and applied SE engineering practices in ML development deepen.

Isdahl et al. [24] surveyed ML platforms' support for reproducibility of empirical results. Several platforms considered in the study falls under the ML experiment management systems—which is also the focus of our study. The authors proposed a method to assess ML platforms' reproducibility and analyzed the features which improve their support. Ferenc et al. [27] carried out a comparison of ML frameworks' features, investigating support for features that include data versioning, graphical dashboards, model versioning, and ML workflow support. Weißgerber et al. [28] investigate 40 ML open source platforms for support of full-stack ML research with open science at its core. The authors developed an open science-centered process model that integrates transparency and openness for ML research. The authors found 11 tools and platforms to be most central to the research process. They further analyzed them for resource management and model development capabilities.

Similar to our work, these previous studies have considered tools such as StudioML, MLFlow, Weights and Biases, Polyaxon, Comet.ml, Sacred, Sumatra, and DVC. In contrast to our work, they [24], [27], [28] 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 ML domain and has practical implications to users of the considered tools (see the discussion in Section V).

## VIII. Conclusion

This paper discussed asset management as an essential discipline to scale the engineering of ML-based systems and of ML experiments. It also presented a survey of 17 systematically selected tools with management support for ML assets, identifying their common and distinguishing features. We focused on tools addressing commonly reported ML engineering challenges—mainly experiment management, monitoring, and logging. We performed a feature-based analysis and reported our findings using feature models. We identified five top-level features, namely, supported asset types, collection methods, storage methods, supported operations, and integration to other management tools. We found that over half of our subject tools depend on traditional version control systems for asset tracking, and the abstraction level for assets does not often distinguish between important ML asset types, such as models and datasets.

In future work, we intend to explore proposed ML asset management tools from the literature in addition to the ones used in practice. Such a study will provide an overview characterization of the solution space of the ML asset management techniques. We will provide a more elaborate representation that captures the dependencies between assets as observed in the tools. To study usefulness and adequacy for addressing user needs, we plan to perform user studies, investigating the usage of functionalities provided by the tools.

**Acknowledgement.** Wallenberg Academy Sweden.

## References


[1] F. Kumeno, "Sofware engineering challenges for machine learning applications: A literature review," *Intelligent Decision Technologies*, 2020.

[2] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software Engineering Challenges of Deep Learning," in *SEAA*, 2018.

[3] A. Polyzotis, M. A. Zinkevich, S. Whang, and S. Roy, "Data Management Challenges in Production Machine Learning," in *SIGMOD*, 2017.

[4] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *VL/HCC*, 2016.

[5] V. Sridhar, S. Subramanian, D. Arteaga, S. Sundararaman, D. Roselli, and N. Talagala, "Model governance: Reducing the anarchy of production ML," in *USENIX*, 2018, pp. 351–358.

[6] M. Vartak, H. Subramanyam, W. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "Modeldb: a system for machine learning model management," in *HILDA@SIGMOD*. ACM, 2016, p. 14.

[7] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1990.

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

[9] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines*, 2013.

[10] T. Berger, J.-P. Steghöfer, T. Ziadi, J. Robin, and J. Martinez, "The state of adoption and the challenges of systematic variability management in industry," *EMSE*, vol. 25, pp. 1755–1797, 2020.

[11] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Syst. J.*, vol. 45, pp. 621–645, July 2006.

[12] J. Aronsson, P. Lu, D. Strüber, and T. Berger, "A maturity assessment framework for conversational AI development platforms," in *SAC*, 2021.

[13] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh *et al.*, "The State of the Art in Language Workbenches," in *SLE*, 2013.

[14] L. Linsbauer, T. Berger, and P. Grünbacher, "A classification of variation control systems," in *GPCE*, 2017.

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

[16] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: Networked science in machine learning," *SIGKDD Explor. Newsl.*, vol. 15, no. 2, p. 49–60, Jun. 2014.

[17] I. H. Sarker, F. Faruque, U. Hossen, and A. Rahman, "A Survey of Software Development Process Models in Software Engineering," *IJSEA*, vol. 9, no. 11, pp. 55–70, 2015.

[18] R. Wirth, "CRISP-DM : Towards a Standard Process Model for Data Mining," *KDD*, pp. 29–39, 2000.

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

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

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

[22] ISO/IEC/IEEE, "IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999) IEEE Standard for Information Technology—System and Software Life Cycle Processes—Reuse Processes," *IEEE Std 1517-2010 (Revision of IEEE Std 1517-1999)*, pp. 1–51, 2010.

[23] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[24] R. Isdahl and O. E. Gundersen, "Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms," in *eScience*. IEEE, 2019.

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

[26] 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.*, vol. 41, no. 4, pp. 39–45, 2018.

[27] R. Ferenc, T. Viszkok, T. Aladics, J. Jász, and P. Hegedűs, "Deep-water framework: The Swiss army knife of humans working with machine learning models," *SoftwareX*, vol. 12, p. 100551, 2020.

[28] T. Weißgerber and M. Granitzer, "Mapping platforms into a new open science model for machine learning," *it - Information Technology*, vol. 61, no. 4, pp. 197–208, 2019.

[29] S. Idowu, C. Åhlund, and O. Schelén, "Machine learning in district heating system energy optimization," in *PerCom Workshops*. IEEE Computer Society, 2014, pp. 224–227.

[30] S. Idowu, S. Saguna, C. Åhlund, and O. Schelén, "Forecasting heat load for smart district heating systems: A machine learning approach," in *SmartGridComm*, 2015.

[31] S. Idowu, S. Saguna, C. Ahlund, and O. Schelen, "Applied machine learning: Forecasting heat load in district heating system," *Energy and Buildings*, vol. 133, pp. 478–488, dec 2016.

[32] S. Strüder, M. Mukelabai, D. Strüber, and T. Berger, "Feature-oriented defect prediction," in *SPLC*, 2020.

[33] H. Abukwaik, A. Burger, B. K. Andam, and T. Berger, "Semi-automated feature traceability with embedded annotations," in *ICSME*, 2018.

[34] R. Queiroz, T. Berger, and K. Czarnecki, "Towards predicting feature defects in software product lines," in *FOSD*, 2016.

[35] F. Pedregosa, O. Grisel, R. Weiss, A. Passos, M. Brucher, G. Varoquax, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and M. Brucher, "Scikit-learn: Machine Learning in Python," *JMLR*, vol. 12, no. 85, pp. 2825–2830, 2011.

[36] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *OSDI*, 2016.

[37] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, and K. Czarnecki, "What is a feature? a qualitative study of features in industrial software product lines," in *SPLC*, 2015.

[38] R. Tatman, J. Vanderplas, and S. Dane, "A Practical Taxonomy of Reproducibility for Machine Learning Research," *Reproducibility in Machine Learning Workshop @ ICML*, 2018.

[39] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks—a publishing format for reproducible computational workflows," in *ELPUB*, 2016.

[40] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *CHI*, 2020, p. 1–12.

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.