Machine learning experiment management tools: a mixed-methods empirical study

Empirical Software Engineering (2024) 29:74
https://doi.org/10.1007/s10664-024-10444-w

Machine learning experiment management tools: a mixed-methods empirical study

Samuel Idowu[1] · **Osman Osman**[1] · **Daniel Strüber**[1,2] · **Thorsten Berger**[1,3]

Accepted: 3 January 2024

Abstract

Machine Learning (ML) experiment management tools support ML practitioners and software engineers when building intelligent software systems. By managing large numbers of ML experiments comprising many different ML assets, they not only facilitate engineering ML models and ML-enabled systems, but also managing their evolution—for instance, tracing system behavior to concrete experiments when the model performance drifts. However, while ML experiment management tools have become increasingly popular, little is known about their effectiveness in practice, as well as their actual benefits and challenges. We present a mixed-methods empirical study of experiment management tools and the support they provide to users. First, our survey of 81 ML practitioners sought to determine the benefits and challenges of ML experiment management and of the existing tool landscape. Second, a controlled experiment with 15 student developers investigated the effectiveness of ML experiment management tools. We learned that 70% of our survey respondents perform ML experiments using specialized tools, while out of those who do not use such tools, 52% are unaware of experiment management tools or of their benefits. The controlled experiment showed that experiment management tools offer valuable support to users to systematically track and retrieve ML assets. Using ML experiment management tools reduced error rates and increased completion rates. By presenting a user's perspective on experiment management tools, and the first controlled experiment in this area, we hope that our results foster the adoption of these tools in practice, as well as they direct tool builders and researchers to improve the tool landscape overall.

Communicated by: Lei Ma

✉ Samuel Idowu
samuelid@chalmers.se

Osman Osman
oo565004@gmail.com

Daniel Strüber
danstru@chalmers.se

Thorsten Berger
thorsten.berger@rub.de

[1] Chalmers | University of Gothenburg, Gothenburg, Sweden

[2] Radboud University, Nijmegen, Netherlands

[3] Ruhr University Bochum, Bochum, Germany

Published online: 29 May 2024

Keywords Machine learning · Experiment management · Artifacts · Asset management · Tools · ML lifecycle

1 Introduction

In recent years, there has been a significant surge in the development of AI, specifically of machine learning (ML) technology. Companies are now realizing the advantages of using ML models instead of excessive manual programming to provide more efficient solutions (Wuest et al. 2016), which ultimately saves time and money. Consequently, ML models are being increasingly employed across different industries and fields (Jordan and Mitchell 2015; Nayak and Dutta 2017; Miotto et al. 2017; Sharma et al. 2020).

The effective management of ML assets is vital to the success of ML-enabled software systems (Nahar et al. 2022; Idowu et al. 2022a, 2024; Nazir et al. 2024)—similar to how it is essential to manage traditional software engineering assets during development. ML assets are the artifacts used during ML model development (Idowu et al. 2022a, 2021, 2022b), including resource artifacts such as datasets and models; software artifacts such as source code files, computational notebooks, and (hyper)-parameters; experiment metadata; and execution metadata and results, such as performance metrics (Zaharia et al. 2018). Emerging from traditional software engineering, practitioners often employ version control systems (VCS) to track ML experiments assets. However, there is a considerable difference between ML and traditional software assets and how they are used (Janardhanan 2020; Arpteg et al. 2018; Lewis et al. 2021). Since traditional VCS tools were not designed with ML development use cases in mind, they do not offer adequate support to the user for exploring the history of ML projects on the right level of abstraction. Since practitioners often perform hundreds of iterations during ML experiments (Bouthillier and Varoquaux 2020), they need proper tool support to effectively manage the involved assets and their versions. Important use case are, for instance, tracing software behavior back to concrete experiment iterations (e.g., for safety purposes), recognizing model drift, enhancing the explainability of the deployed ML models, or understanding past decisions behind performing a concrete experiment (e.g., why a certain hyperparameter or dataset clustering was performed).

To address these needs when developing ML-enabled software systems, there has recently been a surge of ML *Experiment Management Tools*, such as Neptune.ai (Neptune 2021), MLflow (MLflow 2021), and DVC (DVC 2021). We consider ML experiment management tools as a kind of ML asset management tools (Idowu et al. 2022a, which focus on or provide support for the model prototyping or experimentation stages of model production (Schlegel and Sattler 2022). Experiment management tools are either dedicated tools or are integrated with other asset management tools, including ML lifecycle management, pipeline management, and model management tools (Schlegel and Sattler 2022; Idowu et al. 2022a). In general, ML asset management tools provide support for the development of ML components and AI engineering beyond what is available in traditional software engineering tools. Experiment management tools deal with practical concerns of ML experiments, including versioning, traceability, auditability, reproducibility, and collaboration, by offering relevant operations on ML assets. Among others, these operations allow users to compare different experiment iterations and answer factual questions about assets of ongoing or completed experiments—for example, answering post-experiment questions (see Section 2.1) on the assets linked to a specific model version. The available tools differ in their employed paradigms for key operations, including *asset tracking*—typically either based on APIs or a

command line interface (CLI)—and *querying and retrieving*—typically based on graphical dashboards or a CLI (see Section 2.2).

Recognizing their popularity, recent studies have systematically identified several such tools, provided taxonomies, and compared their features (Quaranta et al. 2021; Idowu et al. 2021; Weber and Hußmann 2022; Schlegel and Sattler 2022). Our interactions with practitioners, especially at an industrial conference, reveal that many have looked into such tools. At the same time, some found them unfit for their way of working—a typical problem for tools. To the best of our knowledge, there are no user-based empirical studies on ML experiment management tools, specifically on their actual benefits, effectiveness, and challenges from the user's perspective. Improving our empirical understanding is essential to improve such tools, providing requirements for researchers, tool builders, and educators.

We have conducted the first empirical study on the impact of ML experiment management tools. Our research delves into the challenges users face while using these tools and how they overcome them. We also explore the benefits of using such tools and the support provided by existing tools.

Our study followed a mixed-methods design. First, we surveyed 81 ML practitioners who have attended industry-focused ML conferences, made recent contributions to ML projects on GitHub, or were relevant practitioners recruited via online freelancing services. The survey aimed to gather their personal opinions about the experiment management tools they used, the limitations of adopting the tools, and the perceived benefits and challenges of ML experimentation (see RQ1–3 in Section 3.1). Second, we conducted a comprehensive six-hour controlled experiment where we guided 15 student developers with a background in software engineering to perform typical supervised ML tasks. We measured the effectiveness of two tools that follow different tracking paradigms (API-based and CLI-based), compared to a baseline scenario of not using any such tool, and to each other.

Our survey and controlled experiment complement each other as different methods to answer our research question on ML experiment management tools. The controlled experiment allowed for an in-depth investigation of the tools' impact on user performance and explored the benefits and challenges of participants' experiences. The practitioner survey provided a broader understanding of the challenges and benefits from a practitioner's perspective. We believe that this mixed-methods design is crucial to obtain valid insights into experiment management tools. Our survey and experiment materials are available in an online appendix (Appendix 2022).

We hope that our study on this increasingly popular class of tools, which are highly relevant for software engineers building ML-enabled systems, provides a basis for researchers and tool builders to improve the tools, making them more effective, and increasing their adoption. Our results support educators who train software engineers for building ML-enabled systems, providing informed recommendations regarding using tools in educating novice ML developers. They help practitioners choose whether and which experiment management tool might be useful in their project. We also hope to initiate a dialogue on the relevance of the tools and whether their claimed benefits will pay off or whether organizations should invest in better processes or training of their developers, among other considerations.

2 Background

We now provide background information on ML workflows, experiments, and experiment management tools.

2.1 ML Workflow & ML Experiments

Similar to traditional software engineering, with stages, such as requirements analysis, design, coding, and testing, ML experiments are performed in well-defined processes (Sarker et al. 2015), such as CRISP-DM (Wirth 2000), KDD (Fayyad et al. 1996), and TDSP (Microsoft 2017), stemming from a data science and data mining context. As shown in Fig. 1, these workflows outline the ML stages of implementing ML-based software systems. They can be summarized into stages of requirements analysis, data-oriented processing, model development, and model operation (Kumeno 2020; Visengeriyeva et al. 2021; Wang et al. 2017). Requirements analysis involves eliciting the model requirements and analyzing available data, while data-oriented stages involve data collection, cleaning, labeling, and feature engineering. Model development includes model design, training, evaluation, and optimization (Arpteg et al. 2018; Visengeriyeva et al. 2021). Model operation includes deploying, monitoring, and controlling in-production models. Figure 1 illustrates multiple feedback loops (indicated by the left-pointing arrows) present in the workflow. These loops demonstrate iterations over sets of the workflow stages for a variable number of times until the process results in the desired outcomes (Arpteg et al. 2018).

In practice, substantial development effort goes into establishing viable ML models through ML experiments and model prototyping (Vartak et al. 2016; Schelter et al. 2018). These are often performed ahead of establishing a production-ready development pipeline. In some settings, multiple practitioners experiment on data features provided through feature stores, aiming to obtain the best-performing models. *ML experiments* consist of multiple incremental iterations performed over the development workflow stages as experiment *runs* or *trials*. The required exploratory and experimentation approaches to ML experiment and model prototyping are the primary factors in the different development nature of ML-enabled systems to traditional SE ones. As shown in Fig. 1, the ML workflow contains a linear progression from requirements analysis to operation stages; however, ML workflows are typically non-linear and include multiple feedback loops (indicated by the upward arrows) (Amershi et al. 2019). These feedback loops reflect the multiple experiment *runs*. We describe a *run* as a one-time cycle through the relevant workflow stages, often resulting in a trained model. Each run employs specific assets' versions (e.g., datasets, hyperparameters, source code) within the solution space of a particular ML task. The solution space includes required assets such as datasets from the application domain presenting relevant features for the learning task, a slice or subset of the initial dataset as training data, learning algorithms, and their (hyper)-parameters. A completed run's outcome often includes a trained model, the model performance measurement based on test data, and obtained predictions from an unseen slice or subset of the initial dataset. To find well-performing models, practitioners rely on multiple

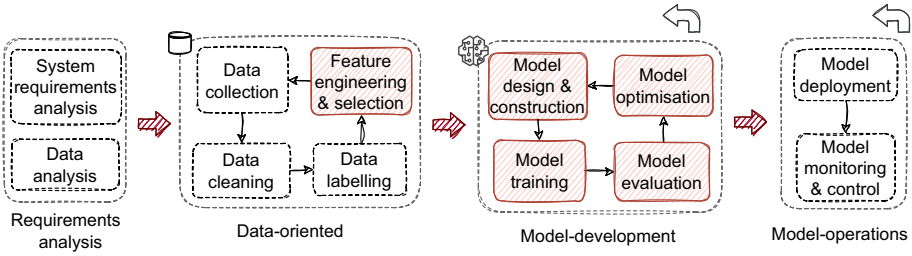

Fig. 1 ML workflow highlighting the stages commonly involved with experimentation

instances of trial-and-error steps, due to the unpredictable nature of ML model performance (Xin et al. 2018; Bouthillier and Varoquaux 2020; Amershi et al. 2019).

Consequently, experiment runs are repeatedly performed while modifying or using new assets until the process results in a model that meets a specific target objective. Such modification includes adding, removing, or engineering features, changing learning algorithms, testing different hyperparameters, and using various performance evaluation metrics. The decision to perform new runs is usually based on an analysis of the results for the current run and its model. Also, during the DevOps-oriented stages (deployment, monitoring, and control of models), there is often a need to modify and make new experiment runs based on newly available data or drift corrections to ensure models stay within the target objective's course. Figure 2 illustrates different asset types modified within a specific run's solution space. Model training involves training datasets, features, learning algorithms, and hyperparameters. Model evaluation involves test datasets, models, predictions, and performance measures. The need to carry out multiple runs is often based on the analysis Y_x of model requirements and resulting model performance M_x when tested with dataset E_x; however, a user may use other requirement metrics to decide if a new run is required(Idowu et al. 2022a).

To find the best-performing combinations of the asset versions over several runs, a manual or automatic approach may be employed. The manual approach follows experts' decisions on necessary step-by-step modifications within the solution space for new runs. The automatic approach-AutoML (Waring et al. 2020; Tuggener et al. 2019; Zhang et al. 2022; Rashidi et al. 2021)- systematically searches a pre-defined portion of the solution space (e.g., a set of hyper-parameters range) for each run. For example, ML models can be automatically selected and parametrized through training loops. Regardless of the employed approach, several experiment runs are often performed before finding optimal models. The data-oriented and model-development stages (highlighted in Fig. 1) involve numerous experiment runs due to the experimental approach often required when designing and building models (Hohman et al. 2020; Bouthillier and Varoquaux 2020). Without dedicated tool support, applied practices may be unstructured and ad hoc, which eventually limits post-experiment tasks and the ability to address experiment concerns effectively. For example, consider an hypothetical developer facing the following problem: *I performed multiple experiment-runs yesterday, and I was able to produce a Root Mean Square Error of 6.5. The problem is that I am not*

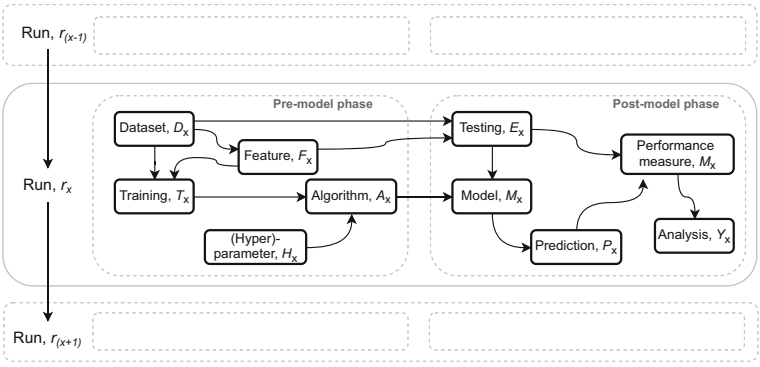

Fig. 2 A representation of an ML experiment run

quite sure which parameters I used, nor do I know which model version I used for that evaluation run. This scenario is prevalent when users have performed so many iterations that it becomes challenging to maintain multiple versions of the involved assets. The need for asset management support is often attributed to the complexity and time overhead that arises with manually managing the large number of asset versions resulting from the multiple exploratory runs (Hill et al. 2016; Vartak et al. 2016; Schelter et al. 2018).

2.2 ML Asset Management & Experiment Management Tools

To provide adequate ML asset management, several tools and platforms support the systematic tracking, collection, storage, and management of ML assets over the various development stages of ML components, their deployment, and integration into software systems. We collectively refer to such systems as *ML asset management tools*. The increasing popularity of such systems implies the growing need for effective asset management for engineering ML-enabled systems (Idowu et al. 2022a). Asset management tools provide various forms of management support, including workflow, pipeline, model, data, and experiment management (Schlegel and Sattler 2022; da Silva et al. 2019).

Experiment management tools treat ML experiment runs as their central abstraction. This category includes dedicated tools, such as Neptune.ai, and multi-purpose tools with other management features, such as MLflow, which provides *MLflow tracking* for experiment management and *MLflow model & registry* for model management. Experiment management tools aim to offer management support during the experimentation and model prototyping stage to reduce the cost, time, and complexities that burden manual or ad hoc asset management. Experiment management tools primarily offer reproducibility and post-experiment analysis for exploratory model development, training, and optimization. The operations offered by experiment management tools complement model development frameworks (e.g., SciKitLearn and TensorFlow (Idowu et al. 2024) and other asset management tools. They primarily offer functionalities to track asset states over multiple experiment *runs* in a structured and organized manner during model development workflow, focusing less on the data-oriented and the DevOps-oriented stages.

Figure 3 illustrates the workflow of using experiment management tools. For the user, they eliminate the risk of forgetting to track or commit important experiment milestones. Most tools capture and record a new version of modified assets when users execute an experiment run—indicating a complete run. For example, DVC offers the command *dvc exp run*, which executes a preconfigured experiment pipeline and simultaneously captures the versions of associated assets. Users can later access prior runs and assets.

We identify the basic tasks offered to users by ML experiment management tools as: i) Tracking assets and; ii) Querying & Retrieving assets (depicted in the Fig. 3). These tasks are fundamental to support the various experiment concerns. *Tracking* involves logging of various asset states when prepossessing data or training models. With the tracking support offered, practitioners can version the diverse asset types used during ML experiments. Additionally, tracking ML assets means that previous experiments can be easily reproduced since all the assets of the experiment were recorded (Sculley et al. 2015). In contrast, the tools support users to *query* and *retrieve* stored assets from previous experiments or runs. The retrieved assets may be reused or retrieved for analysis, such as comparing multiple experiment runs. Assets can also be retrieved from a particular experiment for reuse in a different one. Querying and retrieving assets are essential aspects of post-experiment analysis, where users are often interested in drawing insights from the results of the model development experiments. In a

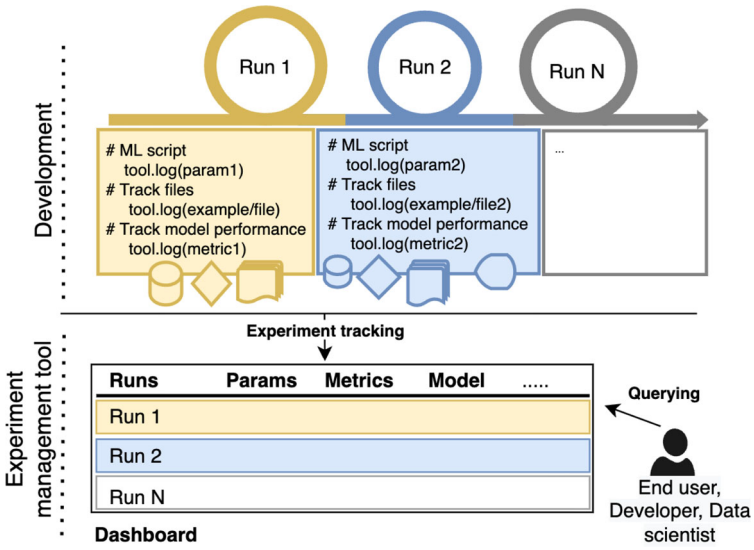

Fig. 3 Workflow of using experiment management tools

related work, we present a meta-model (i.e., a data schema) of the data typically stored in experiment management tools (Idowu et al. 2022b).

The paradigms for tracking assets are either *API*, which requires instrumentation of source code to log state of assets during experiments, or *CLI*, where users pass special commands to track the current states of experiment assets. Similarly, the employed paradigms to query and retrieve stored assets for analysis or other purposes are typically graphical dashboards or CLI. For example, MLflow tracking focuses on capturing, storing, and managing ML artifacts. it provides an API for logging experiment runs, including code and data dependencies, via automatic or manual instrumenting application code. These runs can be viewed, compared, and searched through a Web dashboard UI. DVC, for instance, with the help of its experiment management CLI commands, allows snapshot of all supported assets to be taken for each completed experiment run. Other experiment management tools, such as StudioML, Guild AI, Datmo, and Deepkit, provide similar asset (including source code, dependencies, execution environment, and logs) tracking and querying functionalities to support various experiment concerns such as reproducibility.

In addition, experiment management tools are often available as cloud-based platforms or standalone software tools (Schlegel and Sattler 2022). The cloud-based platform such as Microsoft Azure ML (Azure 2022; Berg 2022), Amazon SageMaker (Amazon SageMaker 2023; Berg 2022), and Google Vertex AI (Vertex ai 2022; Berg 2022) offer multiple ML as a service (MLaaS) tools through the public cloud infrastructure subject to payments. In contrast, standalone software tools such as MLflow (MLflow 2021), Polyaxon (Polyaxon-machine learning at scale 2023), DVC (Control 2023) and Hopsworks (Ormenisan et al. 2020), that can be deployed and used independently in a private computing environment or on-premise.

3 Methodology

We now describe our mixed-methods study design, including survey and experiment design, participant and tool selection, and data analysis. The survey and controlled experiment materials and data are available in our online appendix (Appendix 2022).

3.1 Research Questions

We formulated the following research questions. Since our survey and controlled experiment use a survey questionnaire and an experiment questionnaire (described in Sections 3.2 and 3.3), we already refer to tables with these questions for illustration.

RQ1 *What kinds of experiments are conducted and what experiment management tools, and features, are used?*
With the survey we provide insights into the nature of the experiments of the respondents, then determine what parts of the tool landscape and relevant tool features are considered essential (see Table 1)

RQ2 *What are the perceived benefits of using experiment management tools?*
With the survey we elicit the major reasons why practitioners adopt the use of experiment management tools and their perceived benefits (see Table 2)

RQ3 *What are the challenges and adoption barriers of experiment management tools?*
With the survey we provide insights into the challenges of managing experiments with and without such tools, and possible adoption barriers (see Table 3)

RQ4 *How does the adoption of ML experiment management tools affect user performance?*
With the experiment, we establish whether the tools' assistance is valuable enough to motivate their adoption. We compare the ability of our participants to answer factual questions accurately (see Table 4) when using the management tools versus ad hoc strategies

RQ5 *How are ML experiment management tools, features, and paradigms perceived by users?*
After establishing the value of the experiment management tools, with the experiment, we aim to understand new users' opinions and preferences regarding the paradigms and features of the tools they used in the experiment. We rely on questions eliciting users' views on the usability of the subject tools (see Table 5)

Table 1 Questions on the ML experiment nature and used tools asked in the survey (RQ1)

No.	Question
AQ_1	In terms of ML/DL model training and evaluation, which form of experimentation do you perform?
AQ_2	What is the largest number of experiment runs you have ever performed in a project?
AQ_3	Do you use experiment management tools? If yes, which of the following experiment management tools do you use?
AQ_4	In which ML/DL workflow stages do you use your selected experiment management tool(s)?
AQ_5	Which feature(s) of your experiment management tool(s) do you find important?
AQ_6	Which form/interface of artifact/metadata tracking do you prefer?

multiple choice open-ended

Table 2 Questions on the perceived benefit of tools asked in the survey (RQ2)

No.	Question
BQ_1	To which extent do you agree with the following statements: a) Experiment management tools facilitate my ML/DL tasks well. b) Experiment management tools are easy to learn and use. c) Experiment management tools make me perform experiments more efficiently. d) Experiment management tools improve the performance of my models. e) Overall, they provide a benefit to me compared to not using an experiment management tool. f) A simple command-line interface is sufficient for querying and making analyses of tracked assets. g) A GUI dashboard is essential for querying and analysis of tracked artefacts and metadata. h) I prefer dedicated tools offering strictly experiment management features over multi-purpose tools with additional features.
BQ_2	If applicable, where do you see the benefits/values of the experiment management tool(s) that you use?
	Likert scale multiple choice

3.2 Study Design: Survey

Our survey relied on a questionnaire comprising open-ended, Likert-scale, and multiple-choice questions. Many of the latter provided additional custom fields to ensure unlimited elicitation.

The survey's first part presented a brief introduction to the subject area. We introduced the participants to ML experiment management tools and the importance of the survey and explained example tools in the introduction with an illustration diagram to facilitate comprehension. We also clarified that non-tool users who perform ML experiments could contribute by sharing their experiences as guided by our questionnaire. We asked whether participants had performed ML experiments. We politely terminated the survey for those without such experience. The remainder of this paper refers to the 92.6% who specified having ML exper-

Table 3 Questions on adoption barriers, limitations, and challenges of tools asked in the survey (RQ3)

No.	Question
CQ_1	Are you aware of experiment management tools, such as those mentioned in the previous section? If yes, why are you not using such tools?
CQ_2	How do you manage versions of your experiment artifacts and metadata?
CQ_3	What are the challenges you face in managing artifacts/metadata during or after ML/DL experimentation? Challenges are aspects that make artifact and metadata management difficult.
CQ_4	To which extent do you agree with the following statement: "Specialised experiment management tools can improve artifacts and metadata management during ML/DL experiments/prototyping"
CQ_5	The experiment management tools I use have limitations affecting my experiments
CQ_6	What particular limitations of the tool(s) did you experience?
CQ_7	When using your selected experiment management tool(s), which challenges did you experience? Challenges are aspects that make using the tool difficult.
	Likert scale open-ended multiple choice

Table 4 Factual questions asked post-experiment (RQ4)

No.	Questions
DQ_1	Which *run* performed the best? (i.e., which has the lowest RMSE score?) a) What is the RMSE value for that run?
DQ_2	Which of the algorithms (Linear Regression & Random Forest Regressor) performed best in their first run?
DQ_3	What data features were used for the experimental run with the highest R^2 score?
DQ_4	Compare *Run-4* and *Run-1*. Which one had the highest mean absolute error? a) What was the value?
DQ_5	Compare *Run-5*, *Run-7*, *Run-9* and *Run-11*. Which run had: a) Highest RMSE, b) Highest R^2, c) Highest mean absolute error
DQ_6	If we want to reproduce the results of previous runs, we need to retrieve the model. Which model was used for *Run-4*?
DQ_7	Query Tool for the model with the worst RMSE (Largest value). Provide Run id and the normalized parameter.
DQ_8	Find the runs that produced model evaluation metrics where R^2 is greater than 0.32
DQ_9	List all linear regression runs with RMSE value less than 6.5. Provide Run id and the R^2 value for that run.
DQ_{10}	What is the R^2 value of the very first run. What was the value?
	open-ended multiple choice

Table 5 Perception and opinion questions asked post-experiment (RQ5)

No.	Question
EQ_1	How do you rate the ease of completing the tasks with each tool?
EQ_2	How helpful was the visual dashboard (provided by Neptune), commands (provided by DVC), or the manual approach when comparing the experimental runs
EQ_3	The specialized tools provide significant support for tracking, querying, and retrieving generated data from ML experiments over No-tool.
EQ_4	How long did it take to complete the task for each tool?
EQ_5	Which tool do you consider best for tracking data during machine learning experiments?
EQ_6	Which tool do you consider best for querying and retrieving previously tracked data?
EQ_7	Neptune helps compare different runs using a Web dashboard, while DVC uses CLI. Which do you find most convenient?
EQ_8	Which of DVC and Neptune do you consider the least intrusive in completing the tasks?
EQ_9	Which of DVC and Neptune was easiest to learn?
EQ_{10}	Which tool would you recommend to an ML practitioner?
EQ_{11}	Which of DVC and Neptune provides the best support for comparing different experiment runs?
EQ_{12}	Describe your experience with each of the tools (Neptune, DVC, and No-Tool)
	Likert scale multiple choice open-ended

iment experience as survey participants. We then asked if the participants used specialized experiment management tools. The participants that utilize ML experiment tools provided information used in addressing RQ 1–3, those not using such tools provided information used in addressing part of $RQ3$. The summary of all survey questions is presented in Tables 1 to 3. Lastly, we set questions to help us place participants' responses in the proper context. We asked three questions here-their current role, the number of years of professional experience, and their industries.

3.3 Study Design: Controlled Experiment

Tool Selection Recall that there is a wide range of experiment management tools with varying levels of support, which often extends beyond experiment management alone. From a prior study (Idowu et al. 2021 that systematically identified experiment management tools, we carefully selected two matured and representative example tools with different approaches to tracking, querying, and retrieving ML experiment assets. Specifically, following the two primary paradigms of experiment management tools described in Section 2.2, we chose Neptune.ai, which represents (i) the intrusive API-based paradigm of tracking assets and (ii) the Web dashboard (GUI) paradigm for post-experiment analysis. It is also among the seven most common tools among the 28 identified tools in the survey (excluding custom tools, see Fig. 5).

We chose DVC to represent (i) the CLI-based paradigm of asset tracking and (ii) CLI-based post-experiment analysis. It is also among the eigth most common tools among the 28 identified tools in the survey (excluding custom tools, see Fig. 5). We find it more valuable to compare the tools' paradigms to user support than the tools themselves, since this will make it possible to apply the outcome of this study to other tools. Another motivation is that ML experiment management is a fast-moving space, thus, the subject tools and their principles and paradigms may evolve quickly.

The two tools can be characterized as follows. *Neptune.ai* is a cloud-based service and tool to track ML assets (e.g., datasets, parameters, metrics, and metadata). Tracking relies primarily on developers instrumenting their source code. Assets are tracked as files and metadata, and viewed or explored on a web dashboard for post-experiment analysis. The dashboard allows viewing the experiment runs, their results, and associated assets (Neptune 2021). *DVC* (Data Version Control) is a standalone tool that extends Git to make ML assets (e.g., models and large datasets) shareable. It offers experiment management support through configurable pipelines composed of different stages of the ML workflow. It provides CLI commands to clone and track assets (e.g., models, metrics, and (hyper)-parameters). Fashioned after traditional VCSs, it also provides commands to query or retrieve assets for post-experiment analysis (DVC 2021).

As a baseline for comparison, we consider the *No-Tool* setup, that is, the case of adopting ad hoc strategies without special management assistance from a tool. These include the use of spreadsheet and folder/file naming conventions. This approach is a relevant case to consider, since prior studies show that many practitioners still rely on manual, informal, or ad hoc strategies when managing ML assets (Hill et al. 2016).

Experiment Design We designed a comprehensive experiment to collect participants' experiences using the subject tools and the ad hoc strategies. Our experiment is based on the typical steps when performing supervised ML tasks, such as feature selection and engineering, parameter tuning, and evaluation with different learning algorithms. The highlighted steps in Fig. 1 show the essential activities of our experiments. We asked the participants to perform

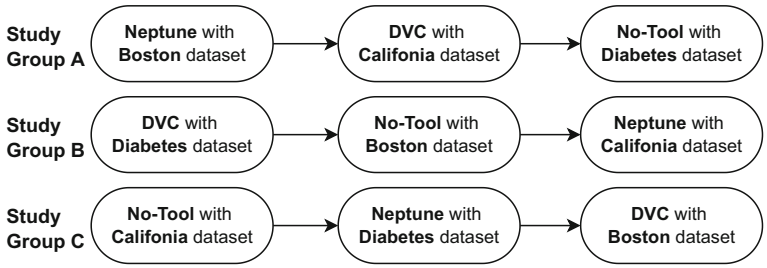

Fig. 4 Cross-over design, varying the tools and datasets

multiple experiment runs of model building by selecting different features, building models with varying data inputs, and evaluating and optimizing the model while tracking the relevant assets. Thereafter, the participants were asked factual questions based on the generated assets during the experiment (see Table 4). To this end, they used the subject tools to query and retrieve specific data from previous runs (a.k.a., post-experiment analysis). To participate in the *No-Tool* setup, users were instructed to refrain from using any experiment management tools and instead utilize their own manual or ad hoc strategies. It was emphasized that tasks should not be repeated and cheating to answer questions was strictly prohibited. In total, the experiment took around six hours per participant.

To improve the validity, we adopted a cross-over design (Lui 2018), dividing our participants into three different study groups with 5 participants per group. For example, participants in group A received treatments in the order of 1, 2, and 3, whereas participants in study group B received treatments in the order of 2, 3 and 1. This design enhances statistical power by abolishing individual subject differences and generating more data points (Turner 2013)—in our case, it increases the number of data points by a factor of 3. Our study groups, sg_A, sg_B, and sg_C, experimented with the same tools and datasets. However, we varied the order of tools and datasets for the groups, as illustrated in Fig. 4. This variation avoids learning effects, as participants cannot infer answers based on previous parts of the experiments.

Variables The *independent* variables in our experiment are the tools—Neptune.ai, DVC, and No-Tool, and the SciKit Learn datasets—Boston, California, and Diabetes. The

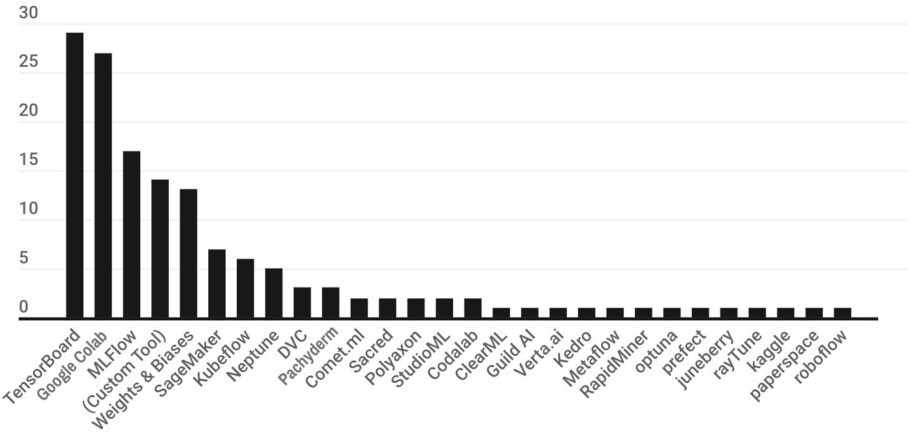

Fig. 5 Used ML experiment management tools (AQ_3)

dependent variables in this experiment are: the error rate and the completion rate of the factual questions posed to the participants (Table 4). The error rate indicates how many wrong answers are provided for each subject tool. The completion rate indicates how many questions were answered for each subject approach. This quantitative data was used to answer $RQ4$. Another dependent variable, used to answer $RQ5$ was the participant's opinion on using the tools and their their usage paradigms. This variable captured various quantitative and qualitative data based on the respective questions in the experiment questionnaire (Table 5).

Experiment Material The material included tutorial documents, an experiment guide with an experiment questionnaire, and Python scripts, all provided via a Google form.

The experiment guide that described the experiment tasks for the participants and contained an experiment questionnaire with questions to be answered during the experiment. We implemented it as a three-part Google form: Part 1 included participant-related questions, such as education level, ML experience, and possible prior experience with related management tools. In Part 2, the participants were presented with ML regression tasks for the Neptune.ai, DVC, and No-Tool setups (with the order varying between groups, as per our cross-over design). We varied among three standard SciKit Learn datasets; Boston, California, and Diabetes datasets (Scikit Learn 2021). For instance, the group sg_A started with Neptune.ai and Boston dataset, while group sg_B started with DVC and Diabetes dataset, see Fig. 4. During the tasks, users were asked to experiment with different regression algorithms (Linear Regression and Random Forest), with different combinations of data features, and different (hyper)-parameters, resulting in multiple experiment runs. We considered a regression task rather than classification, because the model performance metric can be a single numeric value that can be easily interpreted and compared across multiple runs. Following the guided tasks, participants were asked factual questions about their tasks using each tool. We instructed them to use the tools to answer these questions. These questions aimed to investigate how effectively the subject tools support users in retrieving tracked assets by comparing model performances across multiple iterations. We then asked usability questions to elicit the participants' opinions on each tool. In Part 3, we asked general questions about user experience across all the tools and their preferences on tool features.

The scripts for the Python tasks were provided as skeleton scripts. These included important code components like import statements, SciKit Learn code, and pre-filled seed values to ensure a consistent basis for comparison. For the case of DVC, we also provided configuration files accordingly. To give the participants time to set up and familiarize themselves with the subject tools, all participants were given information about the tools 24 hours before the experiment, including setup instructions and a short tutorial.

3.4 Participants

Survey We recruited 81 participants in three different batches. First, practitioners recruited during an industrial ML conference that attracts participants from international top companies working on advanced ML projects (e.g., Spotify, NVidia, Klarna, Volvo, AstraZeneca, and Ericsson). We discussed our research objectives with practitioners, and we followed up with an email invitation to participate in the survey three weeks after the conference. From this batch, we obtained 24 total participants. Second, participants recruited via GitHub, identified by filtering for recent projects with dependencies on the top two ML libraries (Ml-Tooling 2023; Most popular machine learning libraries 2021; Raschka and Mirjalili 2019). We ensured relevant projects by selecting only those invoking the library's methods at least once. After

that, we randomly fetched contributors to projects with commits lower than 60 to potentially obtain users who have worked with model experimentation and are not fully reliant on Git for asset management. Our primary goal is to target users with hands-on experience in machine learning experiments over those who might have worked on large-scale ML projects but not the model experimentation aspects. We sent invitation emails to the contributors and got 25 participants, with a response rate of about 1%. Third, participants recruited via a freelancing service website. To ensure quality, we accepted participation only after reviewing their interested freelancers' profiles and asking controlled questions to establish their qualifications. We accepted participation from roughly 60% of the interested freelancers, giving us 32 participants. The following statistics describe our participants: 35.6% work as data scientists, 31.7% as ML engineers, 12.5% as software engineers, while other indicated roles include data engineers and researchers. The average experience is 4.4 years. 32.2% of the participants indicated technology as their current domain, 17.8% education, 13.6% health, and 11.9% consumer retail. Other indicated domains include consumer retail, telecoms, transport, gaming, and agriculture.

To address the ethical aspects regarding *mining software repositories* (MSR) research activities used for our second batch of participants, we considered guidelines of ethics on MSR that apply to our case. In particular, our MSR activities were solely to recruit relevant survey participants with relevant knowledge and skills and did not include the analysis of research questions based on commit records or code. According to the guidelines by Gold and Krinkle (Gold and Krinke 2022), relevant aspects are those on informed consent, compliance, transparency, and accountability. As described by them, obtaining prior consent from developers contributing to VCS is usually difficult to impossible. This is an intricate topic; unfortunately, there is no consensus on some written guidelines (e.g., IEEE standards). We carefully considered the privacy of contacted users and indicated our legitimate research interests, intention, and the potential benefit of those GitHub users in the long term. To balance user privacy and the need for an adequate number of participants, we sent the invitations successively, in batches. Furthermore, we clearly stated the purpose and benefits of our survey in our invitation letter and sought the participants' consent to collect their opinions.

Controlled Experiment We recruited 15 undergraduate student developers who major in Software Engineering and have taken at least one B.Sc.-level AI/ML course. Since our study elicits the learnability of the subject tools for new users, we considered student developers with a few years of experience as suitable candidates. Several studies (Counsell 2008; Salman et al. 2015; Höst et al. 2000; Berger et al. 2016; Runeson 2003; Falessi et al. 2018) suggest that, in software-engineering-based controlled experiments, students are adequate stand-ins for practitioners, especially when solving tasks with new techniques and tools. Consequently, many empirical software engineering studies have used students as representative stand-ins for practitioners. (Carver et al. 2003; Arisholm et al. 2007; Wels 2012). We applied two selection criteria for our recruitment of participants: (1) Participants must be familiar with ML and how to apply it using popular frameworks, such as SciKit Learn (https://scikit-learn.org/scikit-learn.org) (Idowu et al. 2024). This is vital to avoid programming issues with basic ML concepts, which are outside the scope of our work. (2) Participants must not have prior experience with experiment management tools. These criteria aimed to help us eliminate bias from the outcome of our research. We enforced our selection criteria based on the known student information, and we also confirmed this by asking participants relevant information to confirm they meet our criteria. The incentive for most students stems from mutual benefit: the students were invited to participate in evaluation activities for other students' theses under the premise that the other students would also become evaluation participants for their thesis.

40% of our participants have less than six months of experience with ML, while 60% have over six months of experience with ML.

3.5 Data Analysis

For both survey and the controlled experiment, we obtained a mixture of qualitative and quantitative data from our participants. Two researchers analyzed and reported the results with careful interpretations; then, other authors reviewed the results and the actual responses for consistency.

For the quantitative analysis, we created descriptive statistics for the multiple-choice and Likert-scale answers. For the qualitative analysis of the open-ended questions, we applied thematic analysis. Specifically, identified recurring and essential themes in the participants' responses and organized these themes (a.k.a., codes) in a hierarchy. These coding results are available in our online appendix (Appendix 2022). With the combination of these quantitative and qualitative analyses we answer *RQ1–3* and *RQ5*.

RQ4, which determines the performance of the participants in the different groups (i.e., with the different treatments) in terms in terms of the two dependent variables, *error rate* and *completion rate*, was answered by analyzing the factual questions in the experiment questionnaire. As described above, the variable error rate quantifies how often wrong answers are provided for each subject tool. The completion rate value indicates the extent to which the factual questions were completed for each tool. Upon the results we performed statistical tests. We used Kruskal-Wallis, a non-parametric test for multi-group comparisons, suited for smaller groups that are likely not normally distributed. We applied a Bonferroni correction to the significance threshold of 5% for three comparisons (Neptune vs. DVC, Neptune vs. NoTool, DVC vs. NoTool, explained shortly), leading to a corrected threshold of 1.67%. This analysis will enable us to conclude $RQ4$.

4 Results

We now present the results from our survey (RQ1–3) and our controlled experiment (RQ4–5). On the nature of their ML experiments (AQ_1), 81% of the survey responses indicate manual experiments, where the outputs of each experiment run (model training) were analyzed and evaluated before deciding on the necessary modifications for the next experiment run. In contrast, 58% indicate automated experiments using training loops to find optimal results. The responses show that 41% of participants perform both automated and manual experiments, with 41% and 17% performing only manual and only automated experiments respectively. On the largest count of experiment runs ever performed (AQ_2), 8% of the participants reported having performed 1–10 runs, 24% reported between 10–25 runs, 15% reported 25–50 runs, 31% reported between 50–100 runs, while 23% reported more than 100 runs.

The majority, 69%, of participants in fact use experiment tools (AQ_3). Fig. 5 shows the reported tools. In addition, Table 6 present an overview of the most frequently named tools based on their supported paradigms for asset tracking and querying, which provides context for the choice of tools in our experiment. In particular, tools such as SageMaker, DVC, Pachyderm, Guild AI, and PolyAxon support the CLI approach for tracking and querying. Other popular tools such as TensorBoard, MLFlow, Weights & Biases, Neptune, Comet.ml, and Veta.ai support API-based asset tracking, whereas almost all tools provide a GUI-based approach for asset querying, including the CLI-based ones, which usually let the user choose

Table 6 Characteristics of the top 10 experiment management tools

	Asset tracking mode			Asset querying mode		
	API-based	CLI-based	No support	GUI-based	CLI-based	No support
TensorBoard	✓			✓		
Google colab			✓			✓
MLFlow	✓			✓		
Weights & Biases	✓			✓		
SageMaker	✓	✓		✓	✓	
Kubeflow	✓			✓		
Neptune	✓			✓		
DVC		✓		✓	✓	
Pachyderm		✓		✓	✓	
Comet.ml	✓			✓		

between GUI and CLI. A noteworthy outlier is the second-most named tool, Google Colab, a cloud-based Jupyter notebook environment that provides access to computing resources. For this tool, it is important to note that it is not designed specifically as an experiment management tool and lacks key experiment management features, including built-in solutions for versioning, tracking, querying, or comparing assets from different experiment runs. It does provide integration with experiment management tools that support API-based tracking, such as TensorBoard. A more comprehensive characterization of ML experiment management tools and their features is provided elsewhere (Idowu et al. 2022a, b).

We observed that 70% of those using tools use at least two of them, with an average of 2.7 tools per practitioner. As the essential asset types to manage (AQ_4), 22% of obtained responses state (hyper-)parameter and configuration, 21% model and its metadata, 19% dataset and its metadata, and 18% computation and execution data, including metrics and logs. 13% and 8% consider it necessary to systematically manage "scripts and source code" and "pipeline" assets, respectively. As important features of experiment management tools (AQ_5), 80% of the participants chose visualization, 63% pipeline support, and 51% versioning. Language-agnostic and SaaS features were rated as least important with 18% and

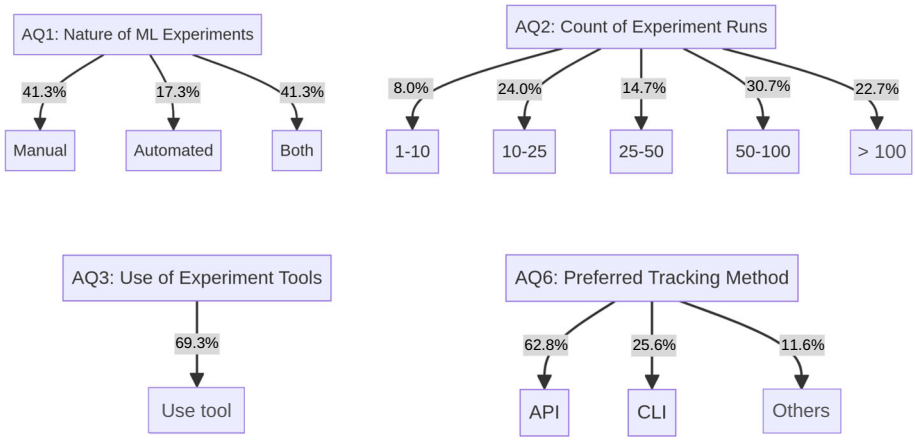

Fig. 6 Nature of experiments, tools, and essential features (AQ_1, AQ_2, AQ_3, AQ_6)

20%, respectively. For the other features, 43% found querying, 41% computational resource provision, 39% VCS integration, and 33% dependency management important. In addition to the multiple-choice options, participants reported compatibility with in-house or custom solutions and test and validation data selection as essential features. Finally, as the preferred usage paradigm for tracking assets (AQ_6), 63% of responses indicate tracking via API in scripts over 26% who preferred the CLI-based approach. Figures 6 and 7 show the summary of RQ1.

Summary

- *While AutoML is increasingly becoming popular, several ML tasks still require manual experimentation, with experiment tasks often requiring up to 100 runs.*
- *The ratio of used experiment management tools per practitioner is about 3 to 1, indicating that the tool landscape is well known as practitioners use multiple experiment management tools.*
- *Practitioners find tracking and managing metadata on experiment assets critical.*

4.1 Perceived Tool Benefits (RQ2)

As shown in Fig. 8, most participants perceived ML experiment management tools to be highly beneficial. 72% of the responses strongly agreed or agreed that tools facilitate their ML tasks ($BQ_{1.a}$), while 18% were neutral. 39% were neutral on the ease of learning and using the tools ($BQ_{1.b}$), while 45% either agreed or strongly agreed to ease of use. 76% strongly agreed or agreed thNature of experiment, tools and essential features (AQ_4, AQ_5)at experiment management tools make them perform experiments efficiently ($BQ_{1.c}$), while 12% were neutral. 48% agreed or strongly agreed that using experiment management tools helps improve their model performance ($BQ_{1.d}$), while 30% were neutral. 74% agreed or strongly agreed they obtain management benefits when using the tools compared to when not using them ($BQ_{1.e}$), while 20% were neutral. 33% disagreed that simple command-line interfaces similar to Git are sufficient for querying and making analyses of tracked experiment

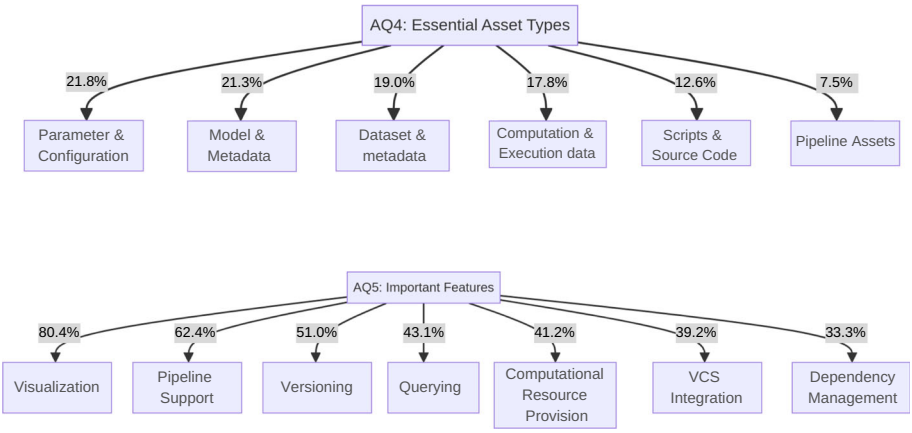

Fig. 7 Nature of experiment, tools and essential features (AQ_4, AQ_5)

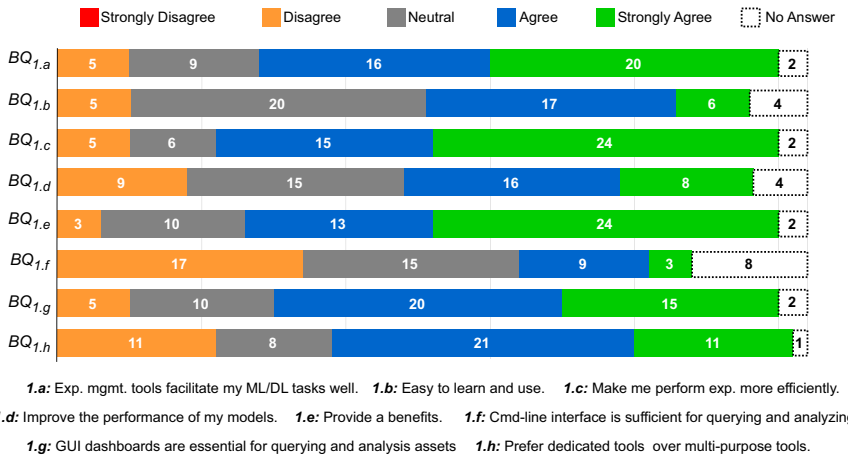

Fig. 8 Result: questions related to perceived benefits (BQ_1)

assets ($BQ_{1.f}$), while 29% were neutral, with 23% that agreed or strongly agreed. 69% agreed or strongly agreed that GUI dashboards are essential for efficient querying and analyses of experiment assets and metadata ($BQ_{1.g}$), while 20% were neutral. 63% prefer or strongly prefer dedicated tools over multi-purpose tools with extended features ($BQ_{1.h}$), 22% do not prefer such, while 16% are neutral.

On the benefits and values of experiment management tools (BQ_2), responses were almost uniform for all the benefits. The benefits, in the order of popularity, are time savings, experiment result analyses and comparison, traceability, reproducibility, result and model optimization, collaboration, and replicability.

Summary

- *Practitioners recognize the benefits of experiment management tools in the following order of importance: experiments' result analysis & comparison, traceability, reproducibility, model optimization, collaboration, and replicability.*
- *Most practitioners prefer dedicated experiment management tools over multi-purpose tools. This is likely because dedicated tools are designed for the specific needs of practitioners and, therefore, offer more specialized and efficient functionality for managing experiments and associated metadata. However, we also noted that GUI-based tools were generally perceived as more beneficial and efficient for asset and metadata management overall, regardless of whether they were dedicated or multi-purpose.*

4.2 Challenges, Adoption Barriers, and Limitations (RQ3)

Experiment Management Without Specialized Tools 52% of the participants who do not use experiment management tools report being aware that such tools exist (CQ_1). However, 37% of them are not using such tools because they lack knowledge or experience. 25% are not using such tools because they prefer tailored or in-house built management tools. 13% of them are not sure they will benefit from using such tools. Other reasons include the extra

cost and time of using such tools, organizational barriers, and data sensitivity levels. When asked how they manage versions of their experiment assets (CQ_2), 44% of them use Git. In contrast, 35% use dedicated naming conventions for folders and files, 13% use custom databases, and 9% do not manage asset versions.

Our participants reported the following challenges they face when *not* using the specialized tools (CQ_3). A challenge is the inability to ensure that essential experiment outputs and their version are consistently and correctly stored, leading to unknowingly overwriting important asses. Another common challenge is the difficulty in retrieving multiple models and corresponding asset versions from previous runs for reuse, especially in projects with many experiment trials. Ad hoc solutions are reported ineffective with increasing experiment runs, making it difficult to track changes made to specific assets over an extended period. In addition, working without specialized tools makes result interpretation difficult due to the lack of visualization to correlate assets to model performance or generate reports to compare different experiment runs. 80% strongly agreed or agreed that specialized experiment management tools can improve asset management, while only 17% were neutral (CQ_4).

Limitation and Challenges With Specialized Tools 6.7% of our participants strongly agreed, and 29.3% agreed, to experience limitations with the tools (CQ_5), affecting their experiments. 50% of the responses were neutral, while 14% either disagreed or strongly disagreed.

The particular issues reported about the tools (CQ_6) are technical restrictions, vendor lock-in, computing resource limitations, missing features, usage costs, and a steep learning curve. Our participants reported various technical issues. For example, 15% of the code count from the thematic analysis indicate tool support for few asset types, while 8% indicate a preference for more flexible and non-restrictive tools with extended support for custom asset types. By design, some tools track assets as immutable objects to ensure persistence. However, some participants indicate this as a limitation. Data accessibility problems were also reported, as some tools do not interface with custom data stores. Our participants also indicated that tools primarily target data scientists and ML engineers and do not fit perfectly into software engineering workflows. Participants have expressed concerns about the limited and simplistic visualization options offered by certain tools. To improve this, we suggest that more customizable visualizations should be made available, including advanced features such as heat maps and network graphs. Additionally, enhancing the usability of these visualization tools would greatly benefit users.

On missing features, participants experienced limitations due to a lack of: automatic parameter search, direct integration with databases, custom ML pipelines support, authorization and authentication support, VCSs (especially Git) integration, and integration with post-deployment operation and existing visualization tools.

Our respondents find cloud service usage cost and computing resources to be limitations. Many tools offer paid cloud-based SaaS, however some offer free services with limited computing resources. As a result, freemium users may experience limitations in terms of storage, memory, and computing resources, while high cost can be a barrier for premium users. Another related limitation is the vendor lock-in issue, which makes it difficult for practitioners to adopt tools or services different from their current vendors. For standalone-tool users, some see the restriction to private computing as a limitation since they are unable to take advantage of faster computing resources. Some participants indicated a steep learning curve as a limitation.

Similarly, regarding challenges experienced when using the tools (CQ_7), 34% of the thematic code count indicate poor documentation or a steep learning curve as a challenge. 14% indicate the tools to lack robustness and consistent availability, making them immature and

buggy. For example, a participant reported experiencing strange tool behavior after reaching hundreds of iterations. 14% indicate challenges in tool setup or usage in development team settings where strong collaboration is required.

Summary

- *Considering practitioners who do not use experiment management tools, the main adoption barrier is a lack of awareness of their benefits.*
- *Practitioners who do not use experiment management tools mainly adopt version control systems and dedicated naming conventions for folders and files to manage multiple runs of experiments.*
- *Practitioners who do not use experiment management tools report that it is challenging to consistently and correctly store or trace experiment assets with alternative approaches.*
- *Considering practitioners who use experiment management tools, the challenges associated with experiment management tools include steep learning curves, robustness, and lack of support for custom setups. Challenges reported for cloud-based tools include vendor lock-in, resource limitations, and high usage costs.*
- *Practitioners who use experiment management tools report missing features as a challenge, indicating that desired tool features are mostly not found in a single tool.*

4.3 User Performance (RQ4)

The error and completion rate for questions DQ_{1-10} in Table 4 reflect the effect of the support offered by the subject tools when performing ML experiments. The tools provide users with the option to organize their experiment assets. For instance, there are greater chances of stating wrong answers to factual questions about completed experiments when there is no structure for organizing assets. To discuss the value of the subject tools, we calculated the error and completion rate for each tool across all study groups (Fig. 9 shows the mean values). The completion rate describes the ratio of attempted questions, while the error rate implies the fraction of wrongly answered to all attempted questions.

The responses for Neptune have an average completion rate of 98% and an average error rate of 7%. DVC obtains an average completion rate of 96% and an average error rate of 29%. The No-Tool alternative has an average completion rate of 84% and error rate of 48%.

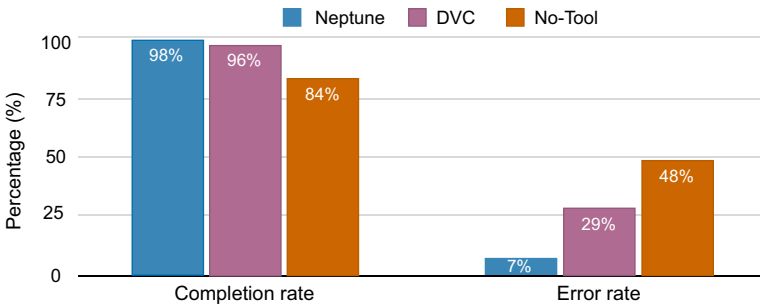

Fig. 9 Results: average completion and error rates

The error rate was lowest when using Neptune, followed by DVC, and participants made the most errors when using the No-Tool alternative. The latter difference vanished in the post-hoc comparison (explained shortly).

To evaluate whether the differences are significant, we conducted a Kruskal-Wallis test together with post-hoc comparisons with a Bonferroni-corrected significant threshold. For the completion rates, we observe one comparison with a p-value smaller than the significance threshold of 0.0167 (Neptune vs. NoTool, p=0.009), and a second case in which p is close to, but not lower than the threshold (DVC vs. NoTool, p=0.04428). We conclude that Neptune differs significantly from No-Tool, while we do not find statistical significance in the other cases. For the error rates, there is a highly significant difference between Neptune vs. NoTool as the p-value is much lower than 0.0167 (p=0.00095). Neptune also shows a significant difference to DVC (p=0.0044). However, the error rates of DVC and No-Tools are not significantly different. Interestingly, the perception of the participants was still different: the overwhelming majority found it difficult to complete the tasks without the use of any ML experiment tool.

Summary

- *The outcome of our controlled experiment established the effectiveness of experiment management tools, as participants could attempt more factual-based questions with lower error rates when using these tools over the manual approach.*
- *The effectiveness of experiment management tools is observed to be higher in the GUI dashboard-based tool than the CLI-based tool, as participants completed more factual-based questions with lower error rates using Neptune than DVC.*

4.4 User Perception on Tools (RQ5)

The responses to the usability questions EQ_{1-12} in Table 5 reflect the users' opinions on and how useful the subject tools are. For the ease of completing the tasks (EQ_1), the overwhelming majority found completing the tasks very easy with Neptune. 73% of the participants found it to be *Easy*, while 20% found it *Very easy*. For the same question about DVC, the responses were slightly positive and mostly neutral, with 46% responding *Neutral*, and 33% responding *Easy*, and 20% responding *Difficult*. Notably, 80% of the participants found using "No-Tool" *Difficult*.

For querying and retrieving assets to compare experimental runs (EQ_2), most participants (93%) found the GUI dashboard of Neptune very helpful, with only 7% neutral responses. When asked the same about DVC's CLI commands, 53% of the participants were neutral, with 47% finding the CLI helpful. For "No-Tool," 53% thought the manual approach was not helpful, with 20% neutral responses.

When asked about the significance of using experiment management tools versus "No-Tool" (UQ_3), all participants agreed (80% *Strongly agree*, 20% *Agree*) the subject tools provide significant support for tracking and retrieving assets during model development. These responses are backed up by the error and completion rates, where the error rates for Neptune and DVC are significantly lower than the No-Tool alternative. Likewise, the completion rates for Neptune and DVC are substantially higher than for No-Tool. Figure 10 summarizes these results.

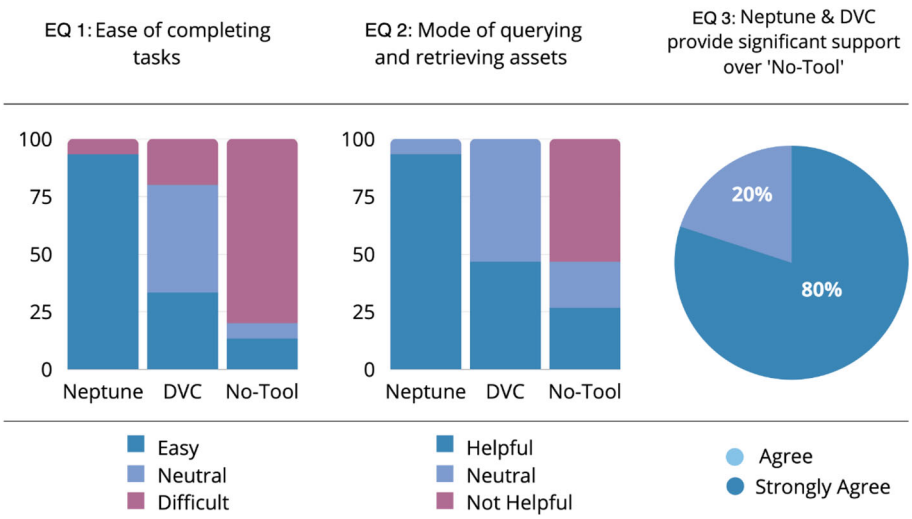

Fig. 10 Results: support for tracking, querying, and retrieving generated data from ML experiments

The average time the participants spent on the experiment tasks varies slightly across the three setups (EQ_4). When using Neptune, the participants spent 1.62 hours on the task. When using DVC, the average time was reduced to 1.5 hours, and they only spent 1.33 hours on the task when using No-Tool. However, it is worth noting that these times are not useful for comparing the time efficiency of the different approaches. This is because many of the reported times come from incomplete attempts at completing the tasks, in particular for the case of No-Tool, in which only 6 out of 15 participants completed all tasks (also see the earlier presentation of completion rates).

When asked about the best tool for tracking assets among Neptune and DVC (EQ_5), the response is balanced, with 53.3% preferring Neptune, and 47% DVC. For the best tool for querying and retrieving previously tracked data (EQ_6), Neptune took the lead with 73% in its favor, while 27% prefer DVC. Also, 73% of the participants prefer Neptune's GUI dashboard for comparison over DVC's CLI commands. When asked about the least intrusive tool (EQ_8), 53% of the participants picked Neptune, while 47% picked DVC. For ease of learning (EQ_9), most participants (73%) believe DVC was the easiest to learn. We believe this reflects the experience of the CLI-based tools, such as Git for managing assets in traditional software engineering. As for the tool to recommend to practitioners (EQ_{10}), 67% said they would recommend Neptune over DVC, while 33% said they would recommend DVC. Lastly, 37% reported that Neptune provides the best support for comparing experiment runs (EQ_{11}). Figure 11 shows the summary of these results.

For the open-ended question EQ_{12}, we report the codes from our thematic analysis. For Neptune.ai, the code "Good UI," referring to responses that mentioned a good user interface (UI), occurs eight times. "Ease in Completing Tasks" comes up three times, indicating answers that mentioned the ease of completing the tasks using Neptune.ai. The "Time-to-Learn" code appears six times, referring to responses that stressed that it took a while to learn how to use the tool. For the DVC responses, the most frequent code was "Simple command," which appeared nine times. This code relates to the answers which commented favorably on simple commands that are easy to understand. Additionally, the "Git" code occurs seven times

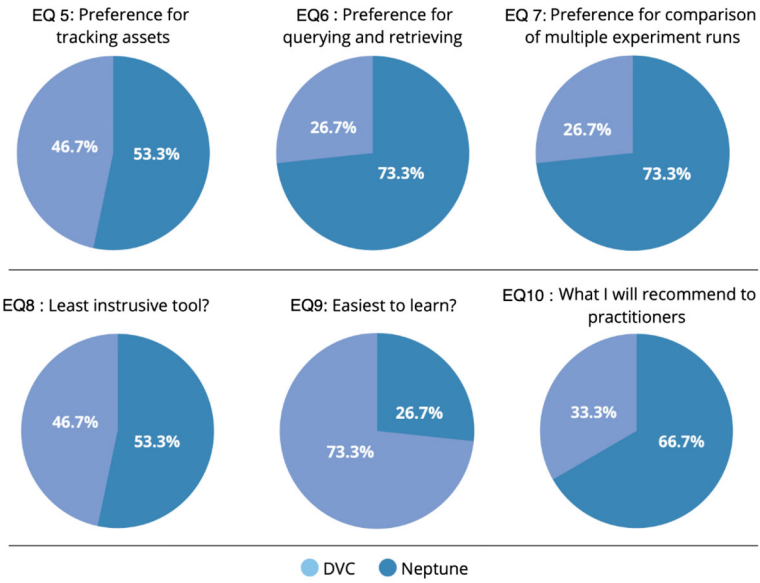

Fig. 11 Results: responses to user opinion questions (Table 5)

in the response: DVC was described as similar to Git, hence making it simple to learn for a user familiar with Git. Consequently, this code was mostly linked with the "Easy-to-learn" code, which occurred six times. There were mainly two codes that occurred in the responses for the No-Tool setup, and that is, the "Time Consuming" and the "Difficult" codes, both occurring at 6 and 8 times, respectively, where "Time Consuming" indicates a long time it took to complete tasks.

One can observe an apparent discrepancy between the average completion times reported earlier and the frequency of the "Time Consuming" code for No-Tool. However, we earlier observed that the reported times partially come from incomplete attempts at completing the tasks. To obtain further insight, we performed a more detailed analysis for the 6 participants whose statements on No-Tool were associated with the code "Time-consuming": 4 of 6 have a lower completion rate for No-Tool than for DVC and Neptune, in one case as low as 38%. In line with this, their textual feedback emphasizes the perceived difficulty and effort of using No-Tool. Of the remaining 2, one provides additional nuance by emphasizing a specific activity—namely, keeping track of values—that they perceived as time-consuming in No-Tool, while the other tools had other time-consuming aspects (e.g., for initial setup). The other remaining participant did not provide relevant details.

Summary

- *Participants found it easier to use GUI-based tools for completing the tasks of managing assets.*
- *Significant numbers of users find the provided support from the specialized tools essential.*
- *The results indicate that there is no clear distinction among user preferences regarding two questions: preference for tracking assets and the tool considered least intrusive.*

5 Discussion

We now discuss the outcomes of our study, their implications, and how they can be used to inform further research on the design and implementation of new features in ML experiment management tools. Recall that our target audience are researchers, tool builders, and educators. With our discussion, we aim to contribute valuable insights and recommendations that can inform the advancement of ML experiment management tools and support the evolving needs of practitioners and researchers in this domain.

Addressing Identified Challenges The survey participants reported various challenges when using ML experiment management tools (See Section 4.2). We make recommendations for the commonly reported limitations and challenges. To improve usability and effectiveness, some actionable results should be considered. First, addressing the *steep learning curves* can be done by providing comprehensive documentation and tutorials. Second, improved user-friendly interfaces that guide users through the tools' functionalities should be considered. Additionally, the maturity of the tools is not too high yet, and maturity should be improved.

To improve *robustness*, the stability and reliability of experiment management tools can be enhanced by conducting rigorous testing, bug fixing, and incorporating error-handling mechanisms. It is essential to consider a broad variety of development use cases during testing, such as large-scale or complex experiments and effective integration with other tools. Supporting *custom setups* can be addressed by offering flexible configurations, customizable workflows, and compatibility with various ML frameworks and libraries. Developers should provide flexible configuration settings, integration capabilities, and extensibility options, allowing users to adapt the tools to their specific needs and integrate them seamlessly with their existing toolchain. For cloud-based tools, tools should offer interoperability and portability features that allow users to easily migrate their experiments and data between different tools platforms or environments. Emphasizing open standards, data portability, and compatibility with popular frameworks can minimize the risk of *vendor lock-in* and provide users with greater flexibility and control. By focusing on these actionable results, developers of ML experiment management tools can enhance practitioners' experience and productivity in managing ML experiments.

Recommendation

– *Enhance ML experiment management tools with easy-to-use interfaces, detailed documentation, and improved learning and robustness. Prioritize compatibility with popular frameworks and allow for customized setups to boost productivity and user experience.*

Integration into Software Engineering Tools In light of the paradigm shift towards ML-enabled systems, it is crucial to develop new, improved, and integrated software engineering tools that can effectively support the unique requirements of ML. Currently, experiment management tools primarily target data scientists and lack interoperability with traditional software engineering (SE) tools. To address this gap, we recommend that new and improved SE tools should incorporate essential experiment management capabilities natively. An example of a step in this direction is the integration of experiment management support in Visual Studio through an extension, as demonstrated by the tool DVC (Dvc extension for visual studio code 2022). Our study results confirmed the positive impact of ML experiment

management capabilities on development performance, with survey participants expressing preferences for the considered paradigms. Therefore, we advocate for the integration of experiment management tools into the ecosystems of traditional software engineering tools, allowing for a seamless and efficient workflow that encompasses both ML and non-ML development activities.

Recommendation

– *Software engineering tools need built-in experiment management features to support ML needs and bridge the gap with data scientists. Integration into existing ecosystems is crucial for a streamlined workflow.*

Tool Paradigms: Balancing Preferences and Recommendations During the survey and experiment, we observed a balanced preference for asset-tracking modalities, namely API-based and CLI-based approaches. This finding contradicted our initial expectations, as API-based tracking is often associated with drawbacks such as manual overhead and increased error-proneness, as discussed in previous studies (Idowu et al. 2021; Ormenisan et al. 2020). One possible explanation for this observation could be the participants' level of experience or familiarity with CLI-based tools. Users who are less comfortable with command-line interfaces may prefer an API-based approach, regardless of the associated overhead. Another explanation could be that the experiment participants did not perceive the additional lines of code required for instrumenting and tracking assets as a significant overhead. Instead, they may have considered it a necessary part of the tasks, given the focus of the experiment on management tools. Since the preferences for asset tracking modalities were evenly balanced, we recommend that future tools support both API- and CLI-based approaches to cater to the varying preferences of users. Furthermore, as part of future work, researchers can explore the specific aspects and components that should be tracked automatically using methods such as Mining Software Repositories (MSR). This investigation can lead to the development of automated methods for tracking assets, improving the efficiency and accuracy of asset management processes. Regarding the comparison between GUI-based and CLI-based paradigms for querying and retrieving, participants found GUI-based tools easier to use, resulting in higher completion rates and fewer errors. This finding aligns with the prevalence of web dashboard interfaces in most tools. However, considering that traditional asset management tools, such as those based on Git, often offer a bimodal interface (CLI and GUI), we recommend that future experiment management tools provide both CLI and GUI paradigms. This approach would effectively cater to users from both software engineering and data science backgrounds, accommodating their distinct preferences and maximizing usability.

Recommendation

– *Future tools should support multiple paradigms for asset management to cater to varying preferences of users, accommodate users from software engineering and data science backgrounds, maximize usability, and address distinct preferences.*

Towards Comprehensive Tools According to findings, practitioners frequently use multiple experiment management tools, with an average of three tools being utilized per practitioner. This highlights our practitioners' familiarity and awareness of the experiment

management tool landscape. However, it remains unknown from our study whether these tools are employed within the same project or if users switch between them over time due to project changes. To gain a deeper understanding, we recommend future research to investigate why practitioners adopt multiple tools. This exploration can provide valuable insights into the tooling landscape, enabling tool builders to better address identified needs and preferences of users. To mitigate the potential impact of complexity arising from heterogeneous development environments caused by multiple tools used concurrently or simultaneously, we propose that experiment management tools be designed as a toolbox with complementary add-ons supporting different use cases and platforms. This approach would foster faster maturity and create a robust ecosystem that can be customized to cater to diverse needs, thereby addressing some of the challenges identified in our study. Additionally, integrating experiment management tools into established traditional software engineering tools, such as integrated development environments (IDEs) commonly used in production-focused development, is recommended. By seamlessly integrating with existing IDEs, these tools would offer easy setup and usage for IDE users, streamlining the adoption process and promoting their widespread use.

Recommendation

– *Create experiment management tools as a toolbox with complementary add-ons to support different platforms and use cases. Integrate them into established software engineering tools for widespread usage.*

Guidance for Educators in Selecting Tools For educators, providing students with the knowledge and skills they need to manage ML experiments effectively is critical. Our study's findings can offer valuable guidance for educators teaching ML-related courses or workshops. When selecting tools to include in educational materials, educators should prioritize usability and user-friendly interfaces. The study revealed that participants preferred tools with intuitive interfaces that guide users through functionalities. Therefore, educators should choose tools that prioritize ease of use and provide comprehensive documentation and tutorials to support students in the learning process. To improve student performance in ML tasks, educators should consider using tools that can help reduce errors and increase completion rates. Our research in Section 4.3 suggests that API-based tools are effective for tracking, while GUI-based tools are useful for querying and retrieving experiment assets. However, based on responses to EQ9 in Section 4.4, we recommend CLI-based tools for students who are already familiar with Git. These guidelines will help students develop the skills needed to manage ML experiments effectively and meet industry demands.

Guidance for Practitioners in Selecting Tools Practitioners must select the appropriate experiment management tools when managing ML projects. To help with this decision, we propose that practitioners assess their project's characteristics and requirements first. Project scale, complexity, team size, available resources, and project goals are all critical considerations in determining the necessity and suitability of experiment management tools. More specifically, we recommend utilizing experiment management tools for large-scale experiments with more than 25 iterations, particularly if done manually rather than through autoML-based experiments. Using specialized tools would be more beneficial for efficiently managing the experiments. We recommend choosing tools based on similar paradigms for practitioners familiar with CLI-based tools. For example, Git users might find DVC easier and more effective than GUI-based tools. We recommend that practitioners select from commonly

used tools, for example, tools with high usage frequency as reported in Fig. 5, as such tools tend to have intuitive interfaces, thorough documentation, and ample learning resources to minimize the learning curve. Based on our result in Section 6, desired functionality and features vary based on tasks and scenarios. For scenarios requiring a lot of experiment data exploration and analysis, we recommend GUI-based dashboard tools that offer visualization out of the box, such as TensorBoard. For scenarios requiring automated experiments, we recommend using tools with pipeline orchestration support, such as MLFlow and Kubeflow. Our findings (EQ7 in Section 4.4) suggest that for scenarios that require a large number of experiment runs, tools with API-based paradigms for asset tracking offer significant benefits. These tools enable seamless integration with automated workflows and assist in managing repetitive experiments. Moreover, based on our results (EQ6 in Section 4.4), we recommend using tools with GUI-based paradigms for querying and retrieving assets for tasks that involve frequent asset retrieval. The user-friendly interfaces and interactive visualizations these tools provide simplify exploring and retrieving experiment assets. They are easy to navigate, allowing practitioners to access assets from multiple iterations efficiently and with minimal errors, thus increasing productivity.

Informal Asset Management In previous studies, it has been observed that practitioners often rely on informal methods, such as notes, spreadsheets, and emails, to track ML assets (Hill et al. 2016). In our study, we found similar patterns, with participants resorting to printing asset values to console output or noting them on paper during the tasks when no systematic tool support was available. However, these informal approaches are known to be costly, time-consuming, and error-prone (Gharibi et al. 2019; Hill et al. 2016). Although some participants were able to answer our factual questions correctly using these informal methods, it is important to note that the completion or error rates for such questions would likely be lower if users were asked after a long period or if the medium used to store the values was not readily accessible. Real-world ML scenarios often involve practical situations that may require accurate answers to factual questions days or even weeks after conducting an experiment. In such cases, a structured and tool-supported approach to asset management becomes crucial. To gain further insights into the impact of informal asset management versus the use of experiment management tools on model development in practical settings, we recommend conducting longitudinal studies that assess the cost-benefit trade-offs associated with the adoption of experiment management tools. These studies would provide valuable insights into the tangible benefits and potential drawbacks of leveraging such tools in real-world ML projects.

Scope and Future Directions The findings presented in this study are based on a selection of ML tasks and scenarios. Our results indicate that GUI-based tools demonstrate advantages for tasks like asset tracking, querying, and retrieving, which were specifically examined in this study. However, it is important to acknowledge that the CLI-based paradigm may have its own strengths and benefits for tasks and scenarios that were not specifically explored here. Therefore, it is recommended that future research explores additional usage scenarios, such as scripting or pipeline integration, to gain a more comprehensive understanding of the capabilities and effectiveness of experiment management tools. While our study focused on specific management tasks, including asset tracking, querying, and retrieving, through a controlled experiment, it is important to recognize that experiment management tools can serve a broader range of functions. For instance, tasks like debugging and model fine-tuning are other critical aspects that can be supported by these tools but were not covered in our study. Thus, for a more comprehensive evaluation and understanding of the potential of experiment management tools, it is recommended that future research explores and investigates their

effectiveness in these additional tasks. This will contribute to a more holistic assessment of the capabilities and usability of such tools in the context of ML asset management.

6 Threats to Validity

External Validity One threat to external validity is the number of participants in our experiment, which was conducted with 15 subjects. In the context of the inherent trade-off between external and internal validity in empirical research (Siegmund et al. 2015), our experiment is leaning towards internal validity, as it provides insights from a six-hour experiment (plus upfront preparation), in which participants interacted with actual tools. As such, it provides much more in-depth insights than, for example, a questionnaire survey can produce—at the price of taking more effort for the participants to complete the experiment, and for us to recruit participants. To enhance validity, our adopted cross-over design, among other benefits (e.g., eliminating individual subject differences as much as possible), maximized the number of data points we could obtain in this setup, leading to 45 observations in total.

Furthermore, our student developers can be considered as practitioners with software engineering experience, but they have only basic knowledge about ML. In fact, prior studies suggest student developers can be representative stand-ins for practitioners *when using tools they are not familiar with* (Counsell 2008; Salman et al. 2015; Höst et al. 2000; Berger et al. 2016; Runeson 2003; Falessi et al. 2018). We confirmed this for our participants. New users are a critical user group because companies might be hesitant to invest in developer tools that require significant specialized experience, and that complicate the onboarding of new employees. They also reflect a large group of software engineers, who develop ML-based systems and have development experience, but only basic knowledge of ML. To enhance the generalizability of our experiment results beyond the two tools considered, we purposely selected them to represent the two broad categories of existing experiment management tools based on a prior study (Idowu et al. 2021). In addition, we report our results over features (which are shared among tools, or could be adopted), not only the concrete tools.

Furthermore, we acknowledge that a group of our participants for our survey were recruited from a specific conference. Even though the conference was attended by professionals from several international companies, recruiting participants from only one conference can still be a source of bias. We strengthened external validity by recruiting participants from two additional sources—GitHub and a freelance service platform—, allowing us to capture a broader range of perspectives and experiences from relevant practitioners.

Internal Validity A threat might be that our survey and experiment tasks and questions reflect our own biases, and bias the participants accordingly. To mitigate this threat, we performed two dry-runs of our survey and experiment from which we sought feedback. Furthermore, we may have misinterpreted our survey and experiment data, which we mitigated by cross-checking all questions, responses, and analysis results by an author not involved in the initial analysis. Finally, in our six-hour experiment, participants might have been subject to fatigue or lack of motivation and have not performed at a consistent level throughout the experiment. We mitigated this threat by counterbalancing: since the tool order was varied between groups, all tools, on average, faced the same order-related advantages and disadvantages. We also allowed the participants to take rest breaks during the experiment. When evaluating the internal validity of this study, it is essential to consider whether the management tasks analyzed in the research cover the entire spectrum of tasks and scenarios that management tools support. It is worth noting that the specific management tasks examined in

this study do not encompass every potential use case or scenario that these tools can handle. As a result, the study's focus on a limited subset of management tasks may restrict its ability to completely capture the breadth and diversity of capabilities provided by management tools. This limitation could affect the generalizability of the study's results to a more extensive range of management tasks and scenarios encountered in real-world situations.

Construct Validity While using a survey gives us the advantage of eliciting information from a large number of participants, it introduces a construct validity threat, where the terminology used in our survey may differ from those the practitioners use or understand. We mitigate this threat by implementing measures to align the terminology and concepts in our survey questionnaire and upfront communication with the participants. First, we provide a summarized explanation of the tools in the invitation letters. Second, the questionnaire instrument starts with an introduction section to aid a common understanding of the survey terms. In addition, we enhance the construct validity of our study by triangulating the responses for specific research questions using both the controlled experiment and the practitioner survey. For instance, our results for RQ2 (in particular, those illustrated in Fig. 8) show that users recognize the benefits of using the tools, confirming our findings from the controlled experiment (RQ4).

A further threat might be hypothesis guessing. Participants might have assumed a hypothesis, e.g., that experiment management tools are better for experiment management and, consequently, might have performed worse when using the 'no-tool' option in our experiments. This threat is mitigated by two factors: First, we asked participants to proceed as they normally would without using experiment management tools. Second, considering the nature of our experiment tasks, the threshold for participants to intentionally adapt their behavior to support an assumed hypothesis - deliberately working slower, making more mistakes and not completing tasks - is arguably higher than for other kinds of tasks (e.g., answering Likert-style questions, which were not asked for the 'no-tool' option). Nevertheless, we cannot rule out that participants might have been demotivated as a result of hypothesis guessing and might therefore have performed worse.

Conclusion Validity To ensure conclusion validity, we employed statistical tests for the tools' performance comparison to mitigate validity threats due to smaller participant groups. In general, we argue that the methodology adopted was appropriate to obtain reliable insights regarding the effect of specialized tools on user performance. In particular, statistical hypothesis tests, such as our employed Kruskal-Wallis test, account for the sample size in a way that ensures robustness: if the sample sizes are small, a stronger difference between the two sets of observations is needed to still be able to conclude significance. For our data this was the case-even though we used a particularly strict correction of the significance threshold for inter-group analysis (Bonferroni). This gives us a high level of confidence in the robustness of our results.

7 Related Work

Prior studies have focused primarily on the features and suitability of these tools for ML practitioners and users (Schlegel and Sattler 2022; Idowu et al. 2022a; Isdahl and Gundersen 2019; Ferenc et al. 2020). However, in contrast to these existing studies, our research targets a different audience: tool developers and researchers who are actively investigating the challenges that arise from extended asset types in the context of developing ML-enabled systems. Our work addresses the unique requirements and considerations faced by these developers,

providing them with valuable insights and guidance for the design and implementation of ML experiment management tools.

As related work, Schlegel and Sattler (2022) conducted a systematic literature review to provide a comprehensive overview of tools, systems, and platforms that facilitate the management of ML assets (Schlegel and Sattler 2022). Their work involved the derivation of assessment criteria, which were subsequently applied to evaluate more than 60 tools across various asset management categories, including experiment management tools. By undertaking this systematic review and assessment, Schlegel et al. contributed valuable insights into the landscape of ML asset management tools, shedding light on the different options available and aiding practitioners in selecting suitable tools for their specific needs. Similarly, to evaluate how research activities towards improved and new ML asset management tools are catching up, Weber et al. (Weber and Hußmann 2022) conducted a systematic literature review, where they analyzed 76 systematically selected relevant publications. They summarized the analyzed tools' trends, strengths, and weaknesses and proposed some potential future directions. Idowu et al. position and discuss *asset management* as an essential discipline to scale the engineering of ML-based systems, facilitating experimenting, development, deployment, and operation of ML-based systems (Idowu et al. 2021, 2022a). The authors survey available tools and present a feature model with common and distinguishing features, such as the supported asset types, the asset collection approach, and their supported operations. Similarly, Quaranta et al. (2021) presented a feature taxonomy of popular experiment management tools.

Serban et al. (2020) analyze academic and grey literature to identify best practices for ML development. They reveal the importance of tracking experiment predictions with model version and input data—a common support operation offered by our considered tools. Isdahl and Gundersen (2019) survey several ML platforms on their support to reproduce empirical results. They propose a new method to assess the subject platforms and analyze features that improve reproducibility. In a similar survey, Ferenc et al. (2020) investigate features such as data versioning, graphical dashboards, model versioning, and ML workflow support available in ML tools. They consider related tools as found in these work (Isdahl and Gundersen 2019; Ferenc et al. 2020), such as DVC and MLflow, as well as several academic prototypes (Tsay et al 2018; Gharibi et al. 2019; Alberti et al. 2018; Schelter et al. 2018; Namaki et al. 2020).

The existing related work primarily targets ML practitioners and generally offers results to inform about the ML tooling landscape, the existing features, and comparisons across different tools (Schlegel and Sattler 2022; Weber and Hußmann 2022; Idowu et al. 2021, 2022a). In contrast, our contributions target tool developers and researchers investigating new ways to improve tooling support for practitioners of AI engineering (Bosch et al. 2022; Bosch 2022; Khomh et al. 2018). Also, in contrast to several related works comparing tools from different asset management categories, we focused on experiment management tools to investigate the value of experiment management support from the user perspective. This is the first work providing empirical-based insight derived from tool users (i.e., ML practitioners) experience and opinion on experiment management tools.

8 Conclusion

We presented a mixed-methods study on ML experiment management tools from the users' perspectives. Our survey with 81 practitioners as well as our controlled experiment with 15 participants shed light on the nature of ML experiments, experiment management tools

benefits, challenges, and adoption barriers, as well as their effects on user performance and perception.

Investigating the nature of ML experiments revealed that manual experimentation still plays a significant role, often involving a substantial number of experiment runs—despite techniques such as AutoML gaining popularity. Additionally, practitioners tend to use multiple experiment management tools, indicating a well-known tool landscape. The tracking and management of metadata on experiment assets were identified as critical aspects for practitioners.

Our practitioners recognized the benefits of experiment management tools for various purposes. These include result analysis and comparison, traceability, reproducibility, model optimization, collaboration, and replicability. It was observed that practitioners generally preferred dedicated experiment management tools over multi-purpose ones. The specialized functionality offered by dedicated tools aligns with the specific needs of practitioners, resulting in more efficient management of experiments and associated metadata. Furthermore, regardless of whether they were dedicated or multi-purpose, GUI-based tools were perceived as more beneficial and efficient for asset and metadata management.

Our study also explored adoption barriers and limitations associated with experiment management tools. Lack of awareness of the benefits of these tools emerged as a significant barrier for practitioners who do not use them. Instead, these practitioners rely on version control systems and naming conventions for managing experiments. However, they face challenges in consistently and correctly storing or tracing experiment assets. On the other hand, practitioners who do use experiment management tools reported challenges such as steep learning curves, robustness issues, lack of support for custom setups, and the absence of desired features. These findings highlight the need for comprehensive tool capabilities and user-friendly interfaces.

Our controlled experiment demonstrated the effectiveness of experiment management tools to enhance user performance. Participants using these tools achieved higher accuracy and completion rates in factual-based questions compared to manual approaches. GUI dashboard-based tools, specifically Neptune, outperformed CLI-based tools like DVC in terms of completion rates and error rates for factual-based questions.

Recall that experiment management tools offer different paradigms to users to interact with them. Our participants found GUI-based tools easier to use for managing assets. Additionally, a significant number of users considered the support provided by specialized tools to be essential. However, the findings also show that there is no clear distinction among user preferences regarding the preference for tracking assets and the tool considered least intrusive.

In conclusion, our study provides valuable insights into the nature of experiments, the perceived benefits of experiment management tools, adoption barriers, limitations, and user performance. The findings highlight the significance of experiment management tools in enhancing efficiency, traceability, and reproducibility in machine learning tasks. The identified challenges and limitations can guide the development of more user-friendly and feature-rich tools. Future research should explore additional usage scenarios beyond the tasks and scenarios considered in this study, investigate the reason behind the use of multiple experiment management tools by a single user, and further evaluate the performance and effectiveness of experiment management tools in different contexts.

Acknowledgements We thank Carl Vågfelt Nihlmar for his valuable contributions during the data collection for the survey presented in this paper. The work was supported by Berger's fellowship granted by the Royal Swedish Academy of Sciences and the Wallenberg Foundation.

Funding Open access funding provided by University of Gothenburg.

Data Availability Statements The experiment and survey response data that support the findings of this study have been deposited in Appendix (2022).

Declarations

Conflicts of interest The authors declared that they have no conflict of interest.

References

Alberti M, Pondenkandath V, Wursch M, Ingold R, Liwicki M (2018) DeepDIVA: a highly-functional python framework for reproducible experiments. ICFHR, pp 423–428

Amazon SageMaker. Available: https://aws.amazon.com/sagemaker/

Amershi S, Begel A, Bird C, DeLine R, Gall H, Kamar E, Nagappan N, Nushi B, Zimmermann T (2019) Software engineering for machine learning: a case study. In: International conference on software engineering: software engineering in practice (ICSE-SEIP). IEEE, pp 291–300

Appendix (2022). Available: https://github.com/isselab/2024-appendix-ml_exp_mgmt_study

Arisholm E, Gallis H, Dyba T, Sjoberg DI (2007) Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Trans Software Eng 33(2):65–86

Arpteg A, Brinne B, Crnkovic-Friis L, Bosch J (2018) Software engineering challenges of deep learning. In SEAA

Azure ai | microsoft cloud (2022). Available: https://azure.microsoft.com/

Berg G (2022) Image classification with machine learning as a service:-a comparison between azure, sagemaker, and vertex ai

Berger T, Völter M, Jensen HP, Dangprasert T, Siegmund J (2016) Efficiency of projectional editing: a controlled experiment. In: FSE, pp 763–774

Bosch J (2022) Introduction to the ai engineering theme. Accelerating Digital Transformation: 10 Years of Software Center, p 399

Bosch J, Olsson HH, Brinne B, Crnkovic I (2022) AI engineering: realizing the potential of AI. IEEE Softw 39(6):23–27

Bouthillier X, Varoquaux G (2020) Survey of machine-learning experimental methods at neurips2019 and iclr2020. Tech, Rep

Carver J, Jaccheri L, Morasca S, Shull F (2003) Issues in using students in empirical studies in software engineering education. In: HealthCom, pp 239–249

Control DV (2023) What is dvc?. Available: https://dvc.org/doc/user-guide/what-is-dvc

Counsell S (2008) Do student developers differ from industrial developers?. In: ITI, pp 477–482

da Silva DN, Simões A, Cardoso C, de Oliveira DE, Rittmeyer JN, Wehmuth K, Lustosa H, Pereira RS, Souto Y, Vignoli LE, Salles R, de Heleno SC, Ziviani A, Ogasawara E, Delicato FC, de Pires PF, da Pinto HLC, Maia L, Porto F (2019) A conceptual vision toward the management of machine learning models. In CEUR Workshop Proceedings 2469:15–27

DVC (2021) Dvc. https://dvc.org/

Dvc extension for visual studio code (2022). Available: https://marketplace.visualstudio.com/items?itemName=Iterative.dvc

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

Fayyad U, Piatetsky-Shapiro G, Smyth P (1996) The KDD process for extracting useful knowledge from volumes of data. Commun ACM 39:27–34

Ferenc R, Viszkok T, Aladics T, Jász J, Hegedüs P (2020) Deep-water framework: the Swiss army knife of humans working with machine learning models. SoftwareX 12:100551
Gharibi G, Walunj V, Rella S, Lee Y (2019) ModelKB: towards automated management of the modeling lifecycle in deep learning. RAISE, pp 28–34
Gold NE, Krinke J (2022) Ethics in the mining of software repositories. Empir Softw Eng 27(1):1–49
Hill C, Bellamy R, Erickson T, Burnett M (2016) Trials and tribulations of developers of intelligent systems: a field study. In VL/HCC, pp 162–170
Hohman F, Wongsuphasawat K, Kery MB, Patel K (2020) Understanding and visualizing data iteration in machine learning. In: Proceedings of the 2020 CHI conference on human factors in computing systems, pp 1–13
Höst M, Regnell B, Wohlin C (2000) Using students as subjects-a comparative study of students and professionals in lead-time impact assessment. ESE 5(3):201–214
Idowu S, Sens Y, Berger T, Krueger J, Vierhauser M (2024) A large-scale study of ml-related python projects. In: 39th ACM/SIGAPP symposium on applied computing (SAC)
Idowu S, Strüber D, Berger T (2021) Asset management in machine learning: a survey. In: ICSE-SEIP. IEEE, pp 51–60
Idowu S, Strüber D, Berger T (2022a) Asset management in machine learning: state-of-research and state-of-practice. ACM Computing Surveys (CSUR)
Idowu S, Strueber D, Berger T (2022b) Emmm: a unified meta-model for tracking machine learning experiments. In: Euromicro conference on software engineering and advanced applications (SEAA)
Isdahl R, Gundersen OE (2019) Out-of-the-box reproducibility: a survey of machine learning platforms. In: eScience. IEEE
Janardhanan P (2020) Project repositories for machine learning with tensorflow. Procedia CS 171:188–196
Jordan MI, Mitchell TM (2015) Machine learning: trends, perspectives, and prospects. Science 349(6245):255–260
Khomh F, Adams B, Cheng J, Fokaefs M, Antoniol G (2018) Software engineering for machine-learning applications: the road ahead. IEEE Softw 35(5):81–84
Kumeno F (2020) Sofware engineering challenges for machine learning applications: a literature review. Intelligent Decision Technologies 13:463–476
Lewis GA, Bellomo S, Ozkaya I (2021) Characterizing and detecting mismatch in machine-learning-enabled systems. In 2021 IEEE/ACM 1st workshop on AI engineering-software engineering for AI (WAIN). IEEE, pp 133–140
Lui KJ (2018) Sample size determination for a 3-treatment 3-period crossover trial in frequency data. Therapeutic innovation & regulatory science 52(4):407–415
Microsoft (2017) Team Data Science Process Documentation. Available: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/
Miotto R, Wang F, Wang S, Jiang X, Dudley JT (2017) Deep learning for healthcare: review, opportunities and challenges. Briefings in Bioinformatics 19(6):1236–1246. Available: https://doi.org/10.1093/bib/bbx044
MLflow (2021) Mlflow. https://mlflow.org/
Ml-Tooling (2023) Ml-tooling/best-of-ml-python: a ranked list of awesome machine learning python libraries. updated weekly. Available: https://github.com/ml-tooling/best-of-ml-python
Most popular machine learning libraries 2014/2021. Available: https://statisticsanddata.org/data/most-popular-machine-learning-libraries
Nahar N, Zhou S, Lewis G, Kästner C (2022) Collaboration challenges in building ml-enabled systems: communication, documentation, engineering, and process. In: Proceedings of the 44th international conference on software engineering, pp 413–425
Namaki MH, Floratou A, Psallidas F, Krishnan S, Agrawal A, Wu Y (2020) Vamsa: tracking provenance in data science scripts
Nayak A, Dutta K (2017) Impacts of machine learning and artificial intelligence on mankind. In: 2017 international conference on intelligent computing and control (I2C2), 2017, pp 1–3
Nazir R, Bucaioni A, Pelliccione P (2024) Architecting ml-enabled systems: challenges, best practices, and design decisions. J Syst Softw 207:111860
Neptune (2021) Neptune.ai. https://neptune.ai/
Ormenisan AA, Ismail M, Haridi S, Dowling J (2020) Implicit Provenance for Machine Learning Artifacts. MLSys'20, p 3
Polyaxon-machine learning at scale. Available: https://polyaxon.com/
Quaranta L, Calefato F, Lanubile F (2021) A taxonomy of tools for reproducible machine learning experiments
Raschka S, Mirjalili V (2019) Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd

Rashidi HH, Tran N, Albahra S, Dang LT (2021) Machine learning in health care and laboratory medicine: general overview of supervised learning and Auto-ML. International Journal of Laboratory Hematology, vol 43, no S1, pp 15–22. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/ijlh.13537
Runeson P (2003) Using students as experiment subjects–an analysis on graduate and freshmen student data. In: EASE, pp 95–102
Salman I, Misirli AT, Juristo N (2015) Are students representatives of professionals in software engineering experiments? In ICSE 1:666–676
Sarker IH, Faruque F, Hossen U, Rahman A (2015) A survey of software development process models in software engineering. IJSEA 9:55–70
Schelter S, Böse JH, Kirschnick J, Klein T, Seufert S (2018) Declarative metadata management: a missing piece in end-to-end machine learning. SysML 2018:3
Schlegel M, Sattler KU (2022) Management of machine learning lifecycle artifacts: a survey. arXiv:2210.11831
Scikit Learn (2021) Datasets: Boston and diabetes. https://scikit-learn.org/stable/datasets/toy_dataset, california, https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html
Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M, Crespo JF, Dennison D (2015) Hidden technical debt in machine learning systems. NIPS 28:2503–2511
Serban A, van der Blom K, Hoos H, Visser J (2020) Adoption and effects of software engineering best practices in machine learning. ESEM
Sharma R, Kamble SS, Gunasekaran A, Kumar V, Kumar A (2020) A systematic literature review on machine learning applications for sustainable agriculture supply chain performance. Computers & Operations Research 119:104926. Available: https://www.sciencedirect.com/science/article/pii/S0305054820300435
Siegmund J, Siegmund N, Apel S (2015) Views on internal and external validity in empirical software engineering. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering. vol. 1. IEEE, pp 9–19
Tsay J, Mummert T, Bobroff N, Braz A, Westerink P (2018) Runway: machine learning model experiment management tool. SysML, pp. 1–3
Tuggener L, Amirian M, Rombach K, Lörwald S, Varlet A, Westermann C, Stadelmann T (2019) Automated machine learning in practice: State of the art and recent results. In: 2019 6th Swiss Conference on Data Science (SDS). pp 31–36
Turner JR (2013) Crossover Design, New York, pp 521
Vartak M, Subramanyam H, Lee WEE, Viswanathan S, Husnoo S, Madden S, Zaharia M (2016) ModelDB: a system for machine learning model management. In the Workshop. ACM Press, pp. 1–3
Vertex ai | google cloud (2022). Available: https://cloud.google.com/vertex-ai
Visengeriyeva L, Kammer A, Bär I, Plöd A (2021) ml-ops.org. Available: https://ml-ops.org/content/end-to-end-ml-workflow
Wang M, Cui Y, Wang X, Xiao S, Jiang J (2017) Machine learning for networking: workflow, advances and opportunities. IEEE Network 32:92–99
Waring J, Lindvall C, Umeton R (2020) Automated machine learning: review of the state-of-the-art and opportunities for healthcare. Artificial Intelligence in Medicine, vol 104, pp 101822. Available: https://www.sciencedirect.com/science/article/pii/S0933365719310437
Weber T, Hußmann H (2022) Tooling for developing data-driven applications: overview and outlook. Proceedings of Mensch und Computer 2022:66–77
Wels S (2012) Test driven development. In: *Proceedings of Agile Seminar 2012*
Wirth R (2000) CRISP-DM: towards a standard process model for data mining. ICKDDM, 24959:29–39
Wuest T, Weimer D, Irgens C, Thoben KD (2016) Machine learning in manufacturing: advantages, challenges, and applications. Production & Manufacturing Research 4(1):23–45
Xin D, Ma L, Liu J, Macke S, Song S, Parameswaran A (2018) Accelerating human-in-the-loop machine learning: challenges and opportunities. In: Proceedings of the second workshop on data management for end-to-end machine learning, ser. DEEM'18. New York, USA: Association for Computing Machinery. Available: https://doi.org/10.1145/3209889.3209897
Zaharia M, Chen A, Davidson A, Ghodsi A, Hong SA, Konwinski A, Murching S, Nykodym T, Ogilvie P, Parkhe M et al (2018) Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull 41(4):39–45
Zhang D, Shen Y, Huang Z, Xie X (2022) Auto machine learning-based modelling and prediction of excavation-induced tunnel displacement. Journal of Rock Mechanics and Geotechnical Engineering, vol 14, no 4, pp 1100–1114. Available: https://www.sciencedirect.com/science/article/pii/S1674775522000786

Samuel Idowu is a PhD computer science and engineering graduate from the University of Gothenburg and Chalmers University of Technology, Sweden. Before that, he completed his Licentiate and M.Sc. degree at Lulea University of Technology, also in Sweden. Samuel's research interests lie in Applied Machine Learning, AI engineering, and empirical software engineering.

Osman Osman is an M.Sc Computer Science and Engineering graduate at the University of Gothenburg.

Daniel Strüber is a Senior Lecturer in Software Engineering at Gothenburg University and Chalmers University of Technology, Sweden, and an Assistant Professor at Radboud University, the Netherlands. After receiving the PhD degree from the University of Marburg in Germany in 2016, he was a Postdoctoral Fellow at the University of Koblenz in Germany and at Gothenburg University and Chalmers University of Technology in Sweden. He is project investigator for competitive grants from the Swedish Research Council and the German Science Foundation. He received six *best-paper* awards. His service was recognized with distinguished reviewer awards at SPLC 2020 and 2023. His research focuses on model-driven software engineering, software product lines, and AI engineering.

Thorsten Berger is a Professor in Computer Science at Ruhr University Bochum in Germany. After receiving the PhD degree from the University of Leipzig in Germany in 2013, he was a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark, and then an Associate Professor jointly at Chalmers University of Technology and the University of Gothenburg in Sweden. He received competitive grants from the Swedish Research Council, the Wallenberg Autonomous Systems Program, Vinnova Sweden (EU ITEA), and the European Union. He is a fellow of the Wallenberg Academy—one of the highest recognitions for researchers in Sweden. He received two *best-paper* and two *most-influential-paper* awards. His service was recognized with distinguished reviewer awards at the tier-one conferences ASE 2018 and ICSE 2020, and at SPLC 2022. His research focuses on model-driven software engineering, program analysis, empirical software engineering, and AI engineering.