On the Effectiveness of Machine Learning Experiment Management Tools

Samuel Idowu[1], Osman Osman[1], Daniel Strüber[1,2], Thorsten Berger[1,3]
[1]Chalmers | Gothenburg University, Sweden, [2]Radboud University, Netherlands, [3]Ruhr University Bochum, Germany

Abstract

Machine learning experiment management tools support developers and data scientists on planning, tracking, and retrieving machine-learning experiments and assets when building intelligent software systems. Among others, they allow tracing back system behavior to experiment runs, for instance, when model performance drifts. Unfortunately, despite a surge of these tools, they are not well integrated with traditional software engineering tooling, and no hard empirical data exists on their effectiveness and value for users. We present a short research agenda and early results towards unified and effective software engineering and experiment management software.

CCS Concepts

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

Keywords

machine learning, experiment management tools, SE4AI

ACM Reference Format:
Samuel Idowu, Osman Osman, Daniel Strüber, Thorsten Berger. 2022. On the Effectiveness of Machine Learning Experiment Management Tools. In *44nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510457.3513084

1 Introduction

Managing development assets is crucial during software engineering. For machine-learning (ML) projects, effectively managing ML assets is similarly important. ML assets include resource artifacts (e.g., datasets and models), software artifacts (e.g., source files), computational notebooks, hyperparameters, as well as experiment and execution results and metadata (e.g., performance metrics [4, 5]). Unfortunately, traditional software engineering tooling—most importantly, version-control systems (VCS)—is not suited to manage ML assets, given the diversity of ML asset types and the exploratory nature of ML experiments, requiring many and hard to predetermine runs (trials) [1, 3, 7]. Since traditional VCS were not designed for ML use-cases, they lack adequate support for exploring the history of ML projects, including queries such as 'which hyperparameters were used in an experiment run where precision was >0.7?'

ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05.
https://doi.org/10.1145/3510457.3513084

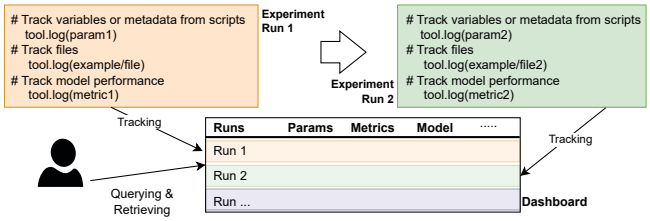

Figure 1: Illustration of experiment management tools

A recent surge of *ML experiment management tools* [5], such as Neptune.ai, MLflow, and DVC, promises enhanced reproducibility, replicability, and collaboration for ML experiments. We identified 17 tools used in practice, plus many research prototypes [4, 5], which facilitate provenance of ML assets and processes by tracking and allowing to explore them. They follow different paradigms, including the *intrusive* paradigm [9], where users instrument source code to track assets, or the *command-line interface* (CLI) paradigm, where tracking is controlled by different commands run by developers, similar to Git. Exploring (i.e., querying and retrieving assets) from experiments relies on GUI (dashboard-like) or CLI paradigms. Figure 1 illustrates the paradigms intrusive and GUI.

Unfortunately, when engineering intelligent, ML-based systems, these tools are not integrated with traditional software engineering tooling. In addition, no evaluations of these tools' effectiveness for users exist—existing works have only compared tool features so far [2, 6, 10]. Our long-term goal is building novel tools supporting the management of software and ML assets in a uniform way. To this end, this paper advocates for empirical assessments of experiment management tools and proposes a research agenda and describes our early empirical results from a controlled experiment. We hope to inspire follow-up work by researchers on assessing, and by tool builders on improving these tools to increase their value for ML and software engineers.

2 Research Goal

The long-term goal is to build the next generation of unified and effective ML experiment management tools integrated with traditional software engineering tooling, such as: IDEs, VCS, VCS hosting platforms (e.g., GitHub), and project management tools. ML experiment management tools have recently gained popularity among the data science tool landscape, supporting the specific (exploratory and iterative) workflows for engineering (e.g., training, evaluating) ML-based components. These tools have first-hand support for the diverse asset types involved. They also serve as alternatives to VCS and hosting platforms for data scientists and ML developers.

Tool Effectiveness. A core research challenge is to determine the effect of tool features and paradigms on users, requiring empirical tool evaluations. Such studies should determine strengths and weaknesses of current tools. Actionable results should provide the basis

to build new or improved experiment management tools suited for user workflows and the diverse asset types.

Unified and Integrated Tools. Another core challenge is integrating and unifying experiment management supports into the traditional software-engineering tooling. To this end, identifying commonalities and differences between the two tool landscapes amond different dimensions, including process, organization, technology, and architecture, is needed. The commonalities can be unified into common tools, while differences remain separate or become add-ons. Results from tool effectiveness studies can further steer the integration, especially which features and paradigms to support and how.

3 Research Agenda and Early Results

Assess Usability and Effectiveness. To elicit hard empirical data on ML experiment management tools, we propose controlled experiments, addressing questions such as: *How do these tools affect user performance? How do users perceive them? How do the different realizations (paradigms) of tool features affect users?* Tool candidates can be sampled from our tool surveys where we identified their features [4, 5], which also allows sampling over tool paradigms. Experiments should have control groups without experiment management tools. Experiments focusing on usability and learnability are also important, incorporating participants with different background and expertise (e.g., experienced vs. non-experienced, practitioner vs. researcher, software engineer vs. data scientist).

Experiments can target specific ML experiment concerns. We identified *tracking*, *querying*, and retrieving as basic concerns operations to all respective tools [5]. Consequently, experiments should exercise these tool operations in the tasks. Participants should be guided through ML tasks that mimic real ML experiments, which involve multiple experiment runs that evolve incrementalls, where participants modify ML assets resulting in new versions; using a specific tool or not. For example, participants are guided through common ML classification tasks, including feature engineering, feature selection, parameter tuning, and model evaluation [1]. Later in the experiment, participants should be required to use the tools' operations to (or manually) track, explore, and retrieve experiment runs and assets.

Independent variables are: (i) user performance and efficiency, and (ii) user perception. The former we propose to measure with the metrics *completion rate* and *error rate* of tasks, as well as *response time* to answer factual questions. These can be elicited with a questionnaire incorporated into the experiment guide. The questions must be designed around common management queries of the experiment assets. For example, participants can be asked to retrieve the algorithm and hyperparameters from a run that resulted in the best-performing model. The independent variable user perception, an essential factor for tool adoption, elicits subjective user opinions on tools and their individual features. Metrics should measure participant ratings on the ease of completing the tasks with each tool, or the level of support offered for tracking, querying, and retrieving ML assets. Results can indicate the essential tool features. In addition, qualitative, open-ended questions complete the picture. Responses might report specific tools or features that are difficult to use, indicating poor usability.

We conducted an experiment where student developers experienced with ML, but without experience with experiment management tools, performed tasks using a GUI-based, a CLI-based, or no tool. Our early results show that the tools offer valuable support in systematically tracking, retrieving, and querying ML assets. They also aid users' performance, reflected in higher completion and lower error rates. Participants preferred GUI-based tooling for post-experiment analysis, while CLI-based tooling was considered easier to learn. Our results indicate that the tools offer valuable support, and that different paradigms can influence the tools' effectiveness.

Compare with Software Engineering Tooling. Studies should determine commonalities and differences between ML experiment management and traditional development tooling, addressing for instance: *What are common features? What are common workflows?* We advocate feature-based surveys, relying on a domain analysis technique, often conducted for similar comparisons, including special kinds of VCS [8]. As an early result, our prior work [4, 5] presents the commonalities and differences of existing ML experiment management tools used in practice. However, studies need to focus on the problem space (required activities and workflows of users) and the design space, architecture, and offered operations.

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

Evaluate Unified Tools. Resulting prototype can to be evaluated with user studies, including experiments, simulations, action research or surveys. Effectiveness and usability are important claims to evaluate, but also scalability and learnability. Unified tools should not compromise compared to the stand-along experiment management tools before. The evaluations can reuse the methods from step 1, but should combine data science and software engineering activities, to evaluate effectiveness of the unification. Users' qualitative perceptions on the new tools can also be elicited through surveys, but we believe action research to be especially fruitful here to understand user interactions and benefits.

References

[1] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *SEAA*.

[2] Rudolf Ferenc, Tamás Viszkok, Tamás Aladics, Judit Jász, and Péter Hegedűs. 2020. Deep-water framework: The Swiss army knife of humans working with machine learning models. *SoftwareX* (2020).

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

[4] Samuel Idowu, Daniel Strüber, and Thorsten Berger. [n.d.]. Asset Management in Machine Learning: State-of-research and State-of-practice. Under revision.

[5] Samuel Idowu, Daniel Strüber, and Thorsten Berger. 2021. Asset Management in Machine Learning: A Survey. In *ICSE-SEIP*.

[6] Richard Isdahl and Odd Erik Gundersen. 2019. Out-of-the-Box Reproducibility: A Survey of Machine Learning Platforms. In *eScience*.

[7] Fumihiro Kumeno. 2020. Sofware engineering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13, 4 (2020), 463–476.

[8] Lukas Linsbauer, Felix Schwaegerl, Thorsten Berger, and Paul Gruenbacher. 2021. Concepts of Variation Control Systems. *Journal of Systems and Software* 171 (2021).

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

[10] Thomas Weißgerber and Michael Granitzer. 2019. Mapping platforms into a new open science model for machine learning. *Information Technology* 61, 4 (2019), 197–208.