MLTEing Models: Negotiating, Evaluating, and Documenting Model and System Qualities

Katherine R. Maffey*
AI Integration Center
U.S. Army
Pittsburgh, PA
kmaffey@alumni.cmu.edu

Kyle Dotterrer*
AI Integration Center
U.S. Army
Pittsburgh, PA
kdottere@alumni.cmu.edu

Jennifer Niemann
AI Integration Center
U.S. Army
Pittsburgh, PA
jmnieman@alumni.cmu.edu

Iain Cruickshank
Army Cyber Institute
United States Military Academy
West Point, NY

Grace A. Lewis
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA

Christian Kästner
Software and Societal Systems Dep.
Carnegie Mellon University
Pittsburgh, PA

***Abstract*—Many organizations seek to ensure that machine learning (ML) and artificial intelligence (AI) systems work as intended in production but currently do not have a cohesive methodology in place to do so. To fill this gap, we propose MLTE (Machine Learning Test and Evaluation, colloquially referred to as "melt"), a framework and implementation to evaluate ML models and systems. The framework compiles state-of-the-art evaluation techniques into an organizational process for interdisciplinary teams, including model developers, software engineers, system owners, and other stakeholders. MLTE tooling supports this process by providing a domain-specific language that teams can use to express model requirements, an infrastructure to define, generate, and collect ML evaluation metrics, and the means to communicate results.**

***Index Terms*—machine learning, test and evaluation, machine learning evaluation, responsible AI**

I. Introduction

Testing is an essential component of product development; in most science and technology domains, consumers assume that a product or piece of equipment is tested for safety considerations before it becomes widely available for use. For instance, US law requires that pharmaceutical companies test drugs before making them available for consumption [1], and the Federal Aviation Administration verifies every aircraft's airworthiness [2], [3]. Despite this norm, there is a lack of accepted best practices for testing *software products that contain machine learning components*, hereafter abbreviated as *ML systems*. As ML gains significant attention in several domains and for many real-world problems, the lack of testing standards becomes more prominent.

Testing ML systems continues to be challenging [4], [5]. While any large software system is complex to develop and deploy [6], the introduction of ML components introduces additional challenges such as defining comprehensive requirements and evaluation criteria to create high-performing models [7]. A lack of rigorous ML evaluation can manifest in negative real-world outcomes. Amazon's gender-biased ML resume vetting tool and the fatal crash caused by a Tesla Model S operating under Autopilot in May 2016 are two examples that demonstrate this reality [8], [9]. As more organizations seek to develop, deploy, and operate ML systems in a growing number of contexts, it is clear that evaluating these systems is important and necessary despite presenting significant challenges [5], [10]–[13].

In this paper, we define *Machine Learning Test and Evaluation* or *MLTE* (pronounced "melt"), a framework and implementation that guide organizations through the ML evaluation process. Based on state-of-the-art ML research, MLTE gives teams a tool to more effectively *negotiate, evaluate, and document* results throughout ML system evaluation. While our focus is ML systems, MLTE is motivated by documented challenges from throughout the ML development life cycle that make defining and evaluating success difficult. Many issues that arise in the ML development process and manifest in production are the result of requirements engineering shortcomings [14], [15]; accordingly, our proposal emphasizes this element of the procedure. To effectively address as many challenges as possible, the MLTE process facilitates evaluation from the inception of a ML project and progresses with it through completion. While many existing ML workflows include evaluation as a step along the path to a finished ML system, we propose that ML system evaluation must be considered throughout the life cycle of a project.

II. Open Problems and State of the Art

Historically, ML research has focused on advancing model capability, typically in terms of accuracy achieved on benchmark problems [16]. Despite the growing adoption of ML components in software products, most education and research still has a model-centric view that rarely considers the system beyond the model [17]. Many practical problems faced by engineering teams happen *at the boundary* between ML components and other parts of the system: examples include unsta-

* Both authors contributed equally to this research.

ble data dependencies and hidden feedback loops [18], limited considerations of system-level qualities like fairness and safety [12], [19], and poor collaboration between data scientists and software engineers [20]. In the following sections, we describe three open problems which are representative of commonly-reported issues throughout the ML development process.

A. Problem 1: Communication Barriers

Research into applications of ML systems and organizational processes used to implement them reveals that communication is a critical yet often neglected part of product development [20], [21]. Organizations often silo team members into distinct software engineering and data science roles; intra-team isolation is exaggerated in organizations that outsource model development. Isolating team members creates communication barriers that often surface as problems during integration when team members realize that particular requirements were not communicated [18], [20]. For example, data scientists may deliver a model that meets accuracy needs but has an inference latency that is unacceptably high for production deployment. Another critical but often ignored aspect of the development process is negotiating requirements for the system and individual components [7], [17], [20], [22]. During model development, teams must consider requirements beyond accuracy such as inference costs, data quality, data quantity, robustness, or fairness requirements [20], [22], [23]. Failing to define and communicate requirements may lead to a mismatch of assumptions from involved entities, which can result in system failures [24].

A final common communication barrier is the process of communicating ML evaluation results amongst teams and to external stakeholders [25]. Tools such as MLflow [26] and Weights and Biases [27] assist model developers with recording experimentation procedures but fail to capture the connection between individual metrics and system outcomes. The *model cards* [28] proposal provides an example of ML results reporting but is intended to document the model rather than the evaluation process it undergoes. Given the existing challenges in sharing evaluation results with users [29], organizations need a method to effectively, consistently, and automatically communicate the results of ML model evaluation.

To address these communication barriers, organizations require a process that establishes consistent communication points, facilitates requirements definition, and offers automated reporting.

B. Problem 2: Low-Quality or Missing Model Documentation

Once model requirements are defined, organizations must ensure that the fulfillment of these requirements is documented appropriately throughout model development. However, requirements documentation remains a challenge for most organizations [20]. Software engineers or system owners must express ML requirements, like a model accuracy or robustness goal, to model developers. In turn, model developers must be capable of providing evidence that these requirements are satisfied. While requirements documentation methods from software engineering are mature, these approaches must be modified to be appropriate for ML systems [21]. Examples of proposals for ML documentation include *datasheets for datasets* [30], *data version control* [31], and *model cards* [28]. However, these proposals focus on models and datasets rather than the requirements they must fulfill, failing to offer teams a documentation method that enforces requirements verification while maintaining comprehensibility.

C. Problem 3: Implementing an Evidence-Based Evaluation

Once requirements are defined and the method by which they are documented is determined, software engineers and model developers must evaluate the model against these requirements and record evaluation results. On one side of this process, software engineers require an expressive way to encode requirements specific to the system under development. On the other side, model developers must translate requirements into measurable ML metrics and capture these measurements as evidence that requirements are satisfied. While most ML libraries include facilities for model evaluation [32], appropriately implementing requirements beyond standard offline accuracy measures on held-out data still entails significant manual effort. A survey of existing research reveals that there is no ML-specific method for specifying and implementing system requirements [15], but examples exist. Breck et al. propose a rubric for quantifying the production readiness of ML systems that includes prescriptions for implementation far beyond just evaluating offline model accuracy [33]; the *deepchecks* project offers a low-level approach to model requirements implementation by providing a library of *checks* that developers can use to evaluate predictive performance [34]. Both of these tools provide useful functionality but fail to fully meet teams' needs because they do not associate the evidence they produce with requirements. Furthermore, they fail to couple a concrete implementation with flexibility sufficient to encompass any ML project. Standardizing and automating model evaluation through test infrastructure is important; organizations benefit from a process and tool that allows them to use selected metrics for model evaluation and export these results as evidence without disruption to the model development process.

In summary, previous work targeting these open problems is generally narrow in scope. Most attempts either design processes for particular aspects of ML model development or introduce a tool addressing specific elements, such as documenting a model's performance under certain data conditions. Our proposal differs by defining a repeatable and measurable process to evaluate ML systems that addresses the entire ML development life cycle.

III. The MLTE Vision

To address the three identified open problems in evaluating ML systems, we propose *Machine Learning Test and Evaluation* (MLTE, pronounced "melt"). MLTE consists of (a) a framework through which organizations evaluate ML models in the context of their intended use and (b) tooling

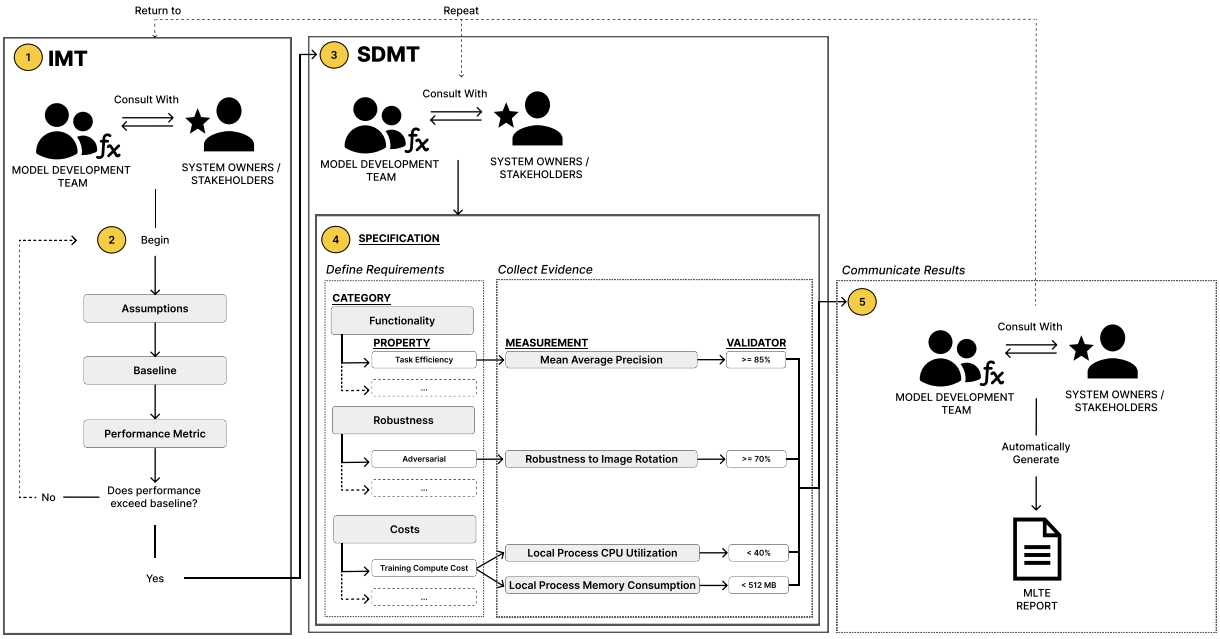

Fig. 1. The MLTE Workflow

that supports the implementation of this framework. MLTE is unique in that it gives organizations a *process* from start to finish that leverages existing state-of-the-art research on testing ML system capabilities as well as interfaces for tracking experiments. MLTE is designed for *interdisciplinary cross-team coordination* and encourages communication by offering specific *collaboration points* and creating *shared artifacts* ("method standardization" and "boundary objects" [35]) throughout the ML development life cycle.

In Figure 1, we outline the MLTE workflow. It starts with ① a negotiation between model developers and the model's consumers (system owners, other stakeholders) regarding model quality requirements; then ② goes through an iterative process of evaluating the model with regard to those requirements until performance exceeds an agreed-upon baseline. Next, ③ model developers and all model and system stakeholders conduct an additional negotiation about model requirements beyond accuracy (e.g., robustness, fairness, inference costs); before ④ evaluating those requirements; finally, ⑤ teams use automated reporting functionality to facilitate communication of their ML evaluation results.

We support each of the steps in the MLTE process with tooling. This tooling is implemented by embedding a domain-specific language for requirements specification and evidence collection in the Python programming language [36] and exposing this functionality via a library. The Python library allows practitioners to define, persist, and share requirements specifications. Furthermore, it provides functionality to generate, persist, and collect artifacts from ML procedures that can be used to verify that requirements are satisfied. Finally, it gives practitioners a mechanism to aggregate this evidence to generate a human-readable report. The MLTE process combined with this tooling provides an adaptive, open infrastructure to guide requirement generation, simplify testing, establish consistency, and communicate ML evaluation results.

IV. Initial MLTE Prototype

To meet its goal of facilitating ML system evaluation, the MLTE prototype consists of preliminary work on both the process and tooling. As described above, the MLTE process contains two primary phases supported by three interspersed collaboration points. In the following section, we detail each of these phases, the manner in which they address our open problems, and how we support each with tooling.[1] MLTE is still in its early stages, and as such we have not yet completed an initial empirical evaluation; our plans to do so are discussed in Section V.

MLTE is built with consideration to related and existing research on ML processes, such as typical ML workflows [37], the ML life cycle [10], CRISP-DM [38], and TDSP [39]. These processes highlight best practices for developing ML systems, examine challenges inherent in bringing a ML system to production, and offer a template for teams to follow as they develop ML products and communicate with their stakeholders. All of these workflows mention requirements as a necessary input for ML development, and all include evaluation as a step in their ML process. Where MLTE diverges from this existing research is in its emphasis on a holistic framework to ensure effective evaluation of ML systems in the context of wider stakeholder and system goals. With its built-in communication points, emphasis on requirements definition, and integration

[1]The MLTE Package is available at https://pypi.org/project/mlte-python, the repository is available at https://github.com/mlte-team/mlte, and the framework is available at https://github.com/mlte-team/mlte-framework.

with programmatic tooling, MLTE gives teams not only a rigorous process but also the tools needed to execute it.

A. Internal Model Testing (IMT)

At the first collaboration point (① in Fig. 1), the model development team and their customers identify model accuracy goals and suitable evaluation metrics as the first step of *Internal Model Testing (IMT)*. The MLTE implementation is structured such that requirements definition directly informs how the model is subsequently evaluated. After completing initial collaboration, teams then proceed to the evaluation portion of IMT (② in Fig. 1).

IMT gives teams an opportunity to refine their ML system implementation plan by conducting team-internal model evaluation. In this step, the MLTE approach balances prescriptiveness with flexibility by providing examples of common evaluations for different types of ML problems, which gives each team sufficient leeway to focus on the relevant concerns of their project. Teams select model characteristics from a curated collection that is organized by ML discipline (e.g., computer vision, natural language processing), and subsequently select a suitable method to assess their chosen characteristics. MLTE prompts teams to evaluate their model with their chosen baseline and performance metric(s), and MLTE tooling supports documentation of this process by providing the functionality to produce and record these results. IMT addresses Problem 2 (Documentation, see Sec. II-B) and Problem 3 (Evaluation, see Sec. II-C) by providing teams with a way to demonstrate that their model meets its requirements, and addresses Problem 1 (Communication, see Sec. II-A) through the collaboration points at ① and ③. An initial exploratory phase in which performance goals evolve is common in ML projects [40]; accordingly, MLTE provides two negotiation points so that teams can make an initial plan for requirements, and then update that plan and add requirements with respect to the system at collaboration point ③. Furthermore, we design IMT to cope with inherent uncertainty [33], [41] by making the process iterative. We envision that teams make changes to their model and execute the IMT process as frequently as needed until the model is ready for *System Dependent Model Testing*.

B. System Dependent Model Testing (SDMT)

The second phase of MLTE, *System Dependent Model Testing (SDMT)* (③ and ④ in Fig. 1), provides a structure and process for evaluating model capabilities and limitations in the context of the *system into which it will be integrated*. The process starts with a collaboration point (③ in Fig. 1), during which model development teams, stakeholders, and system owners describe requirements by defining a *specification*. Teams construct a specification by selecting the characteristics the model must exhibit to be considered acceptable. These characteristics, which we refer to as *properties*, may be any attribute of the trained model, the procedure used to train it (including training data), or its ability to perform inference. A property is 'abstract' in the sense that there may be many ways in which it might be assessed. To assist teams in defining

```python
import mlte
from mlte.specification import Spec
from mlte.property import (
    TaskEfficacy,
    TrainingComputeCost
)
# Initialize the session
mlte.set_model("ModelName", "v0.0.1")
mlte.set_artifact_store_uri("localhost:8080")
# Define the specification
spec = Spec(
    TaskEfficacy(),
    TrainingComputeCost()
)
# Persist the specification to artifact store
spec.save()
```

Listing 1. Defining a *specification* with MLTE.

their requirements, the MLTE framework currently organizes properties into three categories: functionality, robustness, and costs. These categories will expand to include new techniques and categories as research advances. The applicability of the properties within a given category depends on the type of ML task, the objective of the system into which the model is integrated, stakeholder priorities, and value judgments [42]. This process, which occurs at collaboration point ③, addresses Problem 1 (Communication) by establishing a communication waypoint during which system dependent model requirements must be negotiated. Once populated with properties, the specification itself serves as an artifact that addresses Problem 2 (Documentation).

MLTE tooling supports requirements negotiation by providing the ability to programmatically define and persist specifications. Listing 1 shows use of the MLTE library to perform this task. Users import types that represent individual properties and combine instances of these types within a `Spec` object to assemble a new specification. Completed `Spec` objects may then be persisted to an external *artifact store* and subsequently restored for further inspection and manipulation. The ability to programmatically define a specification addresses Problem 3 (Evaluation) by standardizing the process by which it is generated and defining its interface.

Following requirements definition, MLTE prescribes an evidence-based approach to requirements satisfaction (④ in Fig. 1). In this paradigm, model developers determine how the model performs against the requirements set by system owners by collecting *results* that attest to the satisfaction of particular properties. Practitioners produce results by defining and executing *measurements* — functions that assess phenomena related to the property of interest. Model developers define the range of acceptable values for a particular result by defining one or more associated *validators*. A validator is a function that accepts a measurement's output, compares the observed output with an acceptable value or range of values, and produces a result that reflects this comparison. Together, properties, measurements, results, and validators ensure that

```python
# Integrate a result from an external library
from sklearn.metrics import accuracy_score
from mlte.measurement.result import Real
accuracy = Real("accuracy",
    ↪  accuracy_score(...))
# Utilize MLTE-provided measurement
from mlte.measurement.cpu import
    ↪  ProcessCPUUtilization
m = ProcessCPUUtilization("cpu stats")
cpu_stats = m.evaluate(start_training_job())
# Persist results to artifact store
Result.save_all(accuracy, cpu_stats)
```

Listing 2. Generating evidence with internal and external measurements.

```python
# Load the specification
from mlte.specification import Spec
spec = Spec.load()
# Load result(s)
from mlte.measurement.result import Real
accuracy = Real.load("accuracy")
# Validate results, generate report
report = spec.bind(
    {"TaskEfficacy": ["accuracy"]}
    accuracy.greater_than(0.85)
)
report.save()
```

Listing 3. Combining evidence in a *report*.

teams generate evidence to inform larger decisions about the functionality of a system.

This evidence-based approach to requirements satisfaction is labor-intensive, so our programmatic tooling focuses much of its functionality on easing the burden of implementation. To simplify the evaluation process, we provide a small collection of measurement definitions that might not be offered by existing libraries. More importantly, the tooling implements an infrastructure for capturing results produced by any ML library and persisting them in a stable, machine-readable format. Practitioners may utilize existing adaptors defined by the MLTE library to achieve this integration, or they may develop their own to add support for user-defined measurements. Listing 2 demonstrates both aspects of this functionality. Users import internal measurements and evaluate these to produce results, or wrap the output of external measurements in a suitable `Result` type. These `Result` instances are then persisted to the same artifact store that houses completed specifications.

The MLTE process concludes with the generation of a *report* that encapsulates all of the knowledge gained about the model and the system as a consequence of the evaluation process (⑤ in Fig. 1). Report production, demonstrated in Listing 3, ensures that teams have a method through which they can both examine and display the results of their work. Users load previously-saved results from the artifact store, validate these results with provided methods, and combine the output from validation with a specification to create a report. Reports give teams a shared artifact to understand the model, its capabilities, and its context, assisting downstream integration and maintenance activities. This allows teams to avoid barriers to creating shared understanding of results described in Problem 1 (Communication) by giving them a method and infrastructure for communicating their results.

We design the SDMT section of MLTE to support diverse ML projects, regardless of discipline. All elements of the MLTE hierarchy (properties, measurements, results, validators) are collections of functions that can be expanded by MLTE users. Such expansions may be maintained internally (e.g., a team-internal repository of custom measurements) or may be contributed back to the MLTE ecosystem by updating the package itself. Because extensibility is a primary feature, MLTE allows users to rapidly integrate new functionality to suit their specific needs. We envision our existing MLTE tooling as the beginnings of an infrastructure for ML evaluation, rather than as a fully-realized toolkit.

V. Future Plans

Overall, the MLTE framework provides guidance that enforces specific process steps while retaining flexibility. Future research will interrogate this tension between rigor and flexibility. In particular, we plan to study the role of shared machine-readable requirements specifications and reported results as boundary objects [35] between teams. Additionally, MLTE is still in an early stage of usage; the evaluation framework and supporting tooling still need to be evaluated by practitioners. To this end, we will conduct an interview study that addresses two inquiries: how organizations are currently evaluating their ML systems, and how MLTE would function in the context of their organizational processes. Input from this study will inform the refinement of both the framework and tooling as we seek to maximize versatility and capability while maintaining ease-of-use.

Acknowledgements

This work was supported in part by the U.S. Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory under Support Agreement No. USMA21050, the U.S. Army DEVCOM C5ISR Center under Support Agreement No. USMA21056, and the U.S. Air Force Research Laboratory under Support Agreement No. USMA2226. The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Military Academy, United States Army, United States Department of Defense, National Science Foundation, or United States Government.

Lewis' work was funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM23-0017). Kästner's work is supported in part by the National Science Foundation (awards 1813598, 2131477, and 2206859).

References

[1] J. J. Darrow, J. Avorn, and A. S. Kesselheim, "FDA approval and regulation of pharmaceuticals, 1983-2018," *JAMA*, vol. 323, no. 2, pp. 164–176, 2020.

[2] F. De Florio, *Airworthiness: An introduction to aircraft certification and operations*. Butterworth-Heinemann, 2016.

[3] Federal Aviation Administration. (2022, June) Airworthiness certification. [Online]. Available: https://www.faa.gov/aircraft/air_cert/airworthiness_certification

[4] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *arXiv:1606.06565*, 2016.

[5] D. Hendrycks, N. Carlini, J. Schulman, and J. Steinhardt, "Unsolved problems in ML safety," *arXiv:2109.13916*, 2021.

[6] D. Zhang and J. J. Tsai, "Machine learning and software engineering," *Software Quality Journal*, vol. 11, no. 2, pp. 87–119, 2003.

[7] S. Studer, T. B. Bui, C. Drescher, A. Hanuschkin, L. Winkler, S. Peters, and K.-R. Müller, "Towards CRISP-ML (Q): A machine learning process model with quality assurance methodology," *Machine Learning and Knowledge Extraction*, vol. 3, no. 2, pp. 392–413, 2021.

[8] J. Dastin, "Amazon scraps secret AI recruiting tool that showed bias against women," in *Ethics of Data and Analytics*. Auerbach Publications, 2018, pp. 296–299.

[9] V. A. Banks, K. L. Plant, and N. A. Stanton, "Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal Tesla crash on 7th May 2016," *Safety Science*, vol. 108, pp. 278–285, 2018.

[10] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the machine learning lifecycle: Desiderata, methods, and challenges," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–39, 2021.

[11] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software*, vol. 164, p. 110542, 2020.

[12] K. Holstein, J. Wortman Vaughan, H. Daumé III, M. Dudik, and H. Wallach, "Improving fairness in machine learning systems: What do industry practitioners need?" in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–16.

[13] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv:1702.08608*, 2017.

[14] F. Ishikawa and N. Yoshioka, "How do engineers perceive difficulties in engineering of machine-learning systems? - questionnaire survey," in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI))*, 2019, pp. 2–9.

[15] H. Villamizar, T. Escovedo, and M. Kalinowski, "Requirements engineering for machine learning: A systematic mapping study," in *47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021, pp. 29–36.

[16] K. L. Wagstaff, "Machine learning that matters," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1851–1856.

[17] C. Kästner. (2022, Jan) Machine learning in production: From models to systems. [Online]. Available: https://ckaestne.medium.com/machine-learning-in-production-from-models-to-systems-e1422ec7cd65/

[18] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[19] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, and J. Törnqvist, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," *Journal of Automotive Software Engineering*, vol. 1, 2020.

[20] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ML-enabled systems: Communication, documentation, engineering, and process," *Organization*, vol. 1, no. 2, p. 3, 2022.

[21] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Computing Surveys (CSUR)*, Apr 2022.

[22] A. Vogelsang and M. Borg, "Requirements engineering for machine learning: Perspectives from data scientists," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2019, pp. 245–251.

[23] J. Siebert, L. Joeckel, J. Heidrich, K. Nakamichi, K. Ohashi, I. Namba, R. Yamamoto, and M. Aoyama, "Towards guidelines for assessing qualities of machine learning systems," in *Proceedings of the International Conference on the Quality of Information and Communications Technology*. Springer, 2020, pp. 17–31.

[24] G. A. Lewis, S. Bellomo, and I. Ozkaya, "Characterizing and detecting mismatch in machine-learning-enabled systems," in *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 133–140.

[25] B. Rakova, J. Yang, H. Cramer, and R. Chowdhury, "Where responsible AI meets reality: Practitioner perspectives on enablers for shifting organizational practices," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–23, 2021.

[26] M. Zaharia. (2018, June) Introducing MLflow: an open source machine learning platform. [Online]. Available: https://www.databricks.com/blog/2018/06/05/introducing-mlflow-an-open-source-machine-learning-platform.html

[27] Weights & Biases. (2022). [Online]. Available: https://wandb.ai/site

[28] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model cards for model reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019, pp. 220–229.

[29] H. Liu, S. Eksmo, J. Risberg, and R. Hebig, "Emerging and changing tasks in the development process for machine learning systems," in *Proceedings of the International Conference on Software and System Processes*. New York, NY, USA: Association for Computing Machinery, 2020, p. 125–134.

[30] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford, "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86–92, 2021.

[31] R. Kuprieiev, D. Petrov, S. Pachhai, P. Redzynski, C. da Costa-Luis, P. Rowlands, A. Schepanovski, I. Shcheklein, B. Taskaya, J. Orpinel *et al.* (2021) DVC: Data version control—git for data & models. [Online]. Available: https://dvc.org

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in *2017 IEEE International Conference on Big Data*. IEEE, 2017, pp. 1123–1132.

[34] S. Chorev, P. Tannor, D. B. Israel, N. Bressler, I. Gabbay, N. Hutnik, J. Liberman, M. Perlmutter, Y. Romanyshyn, and L. Rokach, "Deepchecks: A library for testing and validating machine learning models and data," *arXiv:2203.08491*, 2022.

[35] S. L. Star and J. R. Griesemer, "Institutional ecology, translations, and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39," *Social Studies of Science*, vol. 19, no. 3, pp. 387–420, 1989.

[36] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[37] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[38] R. Wirth and J. Hipp, "CRISP-DM: Towards a standard process model for data mining," in *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, vol. 1. Manchester, 2000, pp. 29–39.

[39] R. B. Severtson, L. Franks, and G. Ericson. (2017) What is the team data science process? [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/data-science-process/overview/

[40] S. Passi and S. J. Jackson, "Trust in data science: Collaboration, translation, and accountability in corporate data science projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–28, 2018.

[41] I. Ozkaya, "What is really different in engineering AI-enabled systems?" *IEEE Software*, vol. 37, no. 4, pp. 3–6, 2020.

[42] I. D. Raji, A. Smart, R. N. White, M. Mitchell, T. Gebru, B. Hutchinson, J. Smith-Loud, D. Theron, and P. Barnes, "Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing," in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020, pp. 33–44.