

This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.

Author(s): Muiruri, Dennis; Lwakatare, Lucy Ellen; Nurminen, Jukka K.; Mikkonen, Tommi

Title: Practices and Infrastructures for Machine Learning Systems : An Interview Study in Finnish Organizations

Year: 2022

Version: Published version

Copyright: © Authors, 2022

Rights: CC BY 4.0

Rights url: <https://creativecommons.org/licenses/by/4.0/>

Please cite the original version:

Muiruri, D., Lwakatare, L. E., Nurminen, J. K., & Mikkonen, T. (2022). Practices and Infrastructures for Machine Learning Systems : An Interview Study in Finnish Organizations. *Computer*, 55(6), 18-29. <https://doi.org/10.1109/mc.2022.3161161>



Practices and Infrastructures for Machine Learning Systems: An Interview Study in Finnish Organizations

Dennis Muiruri, Lucy Ellen Lwakatare, and Jukka K. Nurminen, University of Helsinki
Tommi Mikkonen, University of Jyväskylä

Using interviews, we investigated the practices and toolchains for machine learning (ML)-enabled systems from 16 organizations across various domains in Finland. We observed some well-established artificial intelligence engineering approaches, but practices and tools are still needed for the testing and monitoring of ML-enabled systems.

Today, artificial intelligence (AI) is incorporated in many real-world software systems and services. However, research on the development, deployment, and maintenance of AI-enabled systems in industrial settings reports this to be a challenging task.^{1,2} Large companies, like Google³ and

Facebook,⁴ often report their development practices and infrastructure for AI solutions that are useful for learning. However, many organizations have yet to adopt and tailor the suggested development practices and infrastructures to narrow the gap from mere prototyping to deploying and producing AI solutions.⁵

Machine learning (ML) is a subset of AI, and its techniques involve the use of high-quality data. ML logic is not explicitly programmed but, rather, learned from data. The

Digital Object Identifier 10.1109/MC.2022.3161161
Date of current version: 3 June 2022

development of industrial ML-enabled software systems involves ML pipelines that consist of several interlocking steps. To support the different steps, end-to-end in one environment, ML platforms, like TensorFlow Extended,³ have been proposed to ensure increased automation across the steps.

Because industrial ML pipelines can be complex, understanding their characteristics is essential. In a large organization like Google, some 3,000 ML pipelines comprising more than 450,000 trained ML models continuously update the models at least seven times a day.⁶ The need to support regular model training and updates in production is a common requirement in most industrial ML-enabled systems because the performance of the models deteriorates over time.¹

Most empirical literature presents the development and maintenance practices of ML-enabled systems from the perspective of a single, often large and experienced online organization. In contrast, we aim to provide empirical evidence of the practices and infrastructure setups across a diverse set of companies in various domains. Through interviews, this study investigated the ML workflow practices and toolchains that are used in the development, deployment, and maintenance of ML-enabled systems in selected organizations in Finland. Our main contributions include

- › empirical evidence of the enacted practices in ML workflows [see the “Practices in ML Workflow (RQ1)” section]
- › the tool adoption in ML pipelines [see the “Tools in ML Pipelines (RQ2)” section] and areas of future research (see the “Implications for Research,” “Implications for Practice,” and “Validity Threats” sections).

BACKGROUND AND RELATED WORK

Software engineering for ML

The adaptation and incorporation of well-established software engineering (SE) methods in the development of ML systems are crucial⁷ because they emphasize other important engineering aspects beyond ML algorithms.¹ With these practices, organizations can address several challenges reported at the different stages of the taxonomy that depict the evolution of ML components in software-intensive systems (experimentation, noncritical deployment, critical deployment, cascading deployment, and autonomous ML components).²

Serban et al.⁵ developed a catalog of 29 SE practices for ML applications based on the literature and, later, measured their adoption rate through a survey with 313 practitioners. The catalog includes SE practices about data (for example, employing sanity checks for all external data sources), training (such as using versioning for data, model, and training scripts), coding (for instance, using continuous integration [CI]), deployment (for example, automating model deployment), team (for instance, collaborating with multidisciplinary team members), and governance (such as enforcing fairness and privacy). Compared to our study, some of the reported SE practices are too general. The lack of details gives room for multiple interpretations, for example, the named practice to use CI. In addition, we study practice enactment by also investigating the toolchains, which Serban et al.⁵ had only speculated would influence the adoption rate of specific practices.

ML workflows and pipelines

ML workflows describe the different tasks performed to develop, deploy,

and operate ML models.⁷ ML pipelines express complex input-output (I/O) relationships between different tasks/operators of an automated ML workflow.⁶ Generally, ML pipelines plug together several tools to automate ML workflows.⁸

A typical ML workflow lifecycle includes model requirements, data collection, cleaning, labeling, feature engineering, model training, evaluation, deployment, and monitoring.⁷ Studies show that the end-to-end automation of ML workflows improves the quality, traceability, development time, and deployment rate of ML models.^{6,8} Furthermore, it allows organizations to reuse common workflow steps across multiple ML systems.^{3,8}

Few studies report the characteristics of ML pipelines in terms of their components, architectures, and tools.^{6,8} Unlike our qualitative analysis, Xin et al.⁶ quantitatively analyzed more than 3,000 ML pipelines at Google and presented their high-level characteristic concerning the pipeline lifespan, complexity, and resource consumption. For the complexity of ML pipelines, the authors analyzed the typical input data shape, feature transformation, and model diversity. The model diversity showed that a large portion were neural networks (NNs) (64%), and the model type and architecture influence the characteristics of the resulting ML pipelines. Xin et al.⁶ identified data management-related areas as key for optimizing ML pipelines.

METHODOLOGY

An exploratory multiple-case study⁹ was conducted between March and August 2021. The research method was selected to gain a deep understanding of the enacted practices and tool support for ML systems in real-world

settings. The main research questions (RQs) include the following:

- › RQ1: What practices are applied in the development, deployment, and maintenance of industrial ML-enabled software systems?
- › RQ2: What tools are used to support the development, deployment, and maintenance of industrial ML-enabled software systems?

Research design and case selection

The main goal of our study was to understand the state of the practice of ML-enabled systems' development and toolchains within the Finnish context. In this study, a case (Table 1) is an organization in Finland with experience in developing, deploying, and maintaining ML-enabled software systems. The main criterion for case selection was that the ML-enabled software system must be operational in a production environment.

We first identified relevant practitioners from different organizations and adapted an interview guide used in earlier research.² For the current study, we modified the questions under the "Project Background and Characteristics" section of the interview guide to inquire only about operational ML systems rather than practitioners' general experiences in ML projects. This was done to exclude ML systems in the experimental stage because they often have immature practices and toolchains.² In addition, we added new questions to the interview guide that inquired about the infrastructure and tools used. The identified practitioners (or their organizations) were primarily known to be working on ML solutions by the researchers, and others were gathered from LinkedIn.

We reached out to 37 organizations via email, out of which 16 agreed to participate in the study. Generally, practitioners were free to choose whether to (and who would) participate, but the researchers purposefully ensured that the organizations were varied in terms of the sector and size. Interviewed practitioners had varying roles, as seen in Table 1. Academically, 10 (43%) practitioners hold a Ph.D. degree, 10 (43%) hold a master's degree, and two (9%) have a bachelor's degree. Ten cases are large organizations, and five constitute small or medium-sized organizations with revenues below €50 million based on 2020 financial reports.

Data collection

Research data were primarily collected through semistructured interviews conducted by two researchers. A total of 23 practitioners from the 16 organizations were interviewed between 24 March and 27 April 2021. All interviews were conducted virtually due to COVID-19. Each interview session took, on average, 80 min.

During the interview session, the researchers presented the details of the study and requested consent to record the interview. One researcher asked the question outlined in the interview guide that contained five broad categories: data management, model training, model deployment, model monitoring, and general challenges. The interview guide was not strictly followed to allow probing questions depending on the interviewees' responses and expertise. All recorded interviews were automatically transcribed using Otter (<https://otter.ai>), and the researchers manually corrected errors in the transcriptions.

Data analysis

The analysis of the interview transcripts mainly consisted of two coding steps

and a session to discuss and harmonize the codes.⁹ A deductive approach formed the first stage of our coding process in which main themes informed by the structure of our interviews were outlined. The themes constituted the high-level codes in our analysis, and these included role and responsibility, organization, ML use case, practices, challenges, and tools. The actual coding of data was done in an iterative manner using both deductive and inductive⁹ approaches within each group and applied broadly at a paragraph or statement level. The subgroups were further refined during researcher meetings.

PRACTICES IN ML WORKFLOW (RQ1)

This section presents common practices observed across the cases, as summarized in Table 2.

Data management

Data collection and storage. Data collection and storage are handled in various ways: batch loading data from internal systems, streaming from devices/sensors, or extracting from third-party vendor application programming interfaces (APIs) or open source repositories. The training data are then commonly stored in cloud platforms, as shown from data sources in Table 1.

Low-level metrics, such as I/O operations per second are considered when choosing a storage architecture; data fetching can constitute a sizeable amount of the overall model training time. Case E uses a mounted discs solution instead of a network drive accessed via a web interface.

Data storage formats. These are factored in when considering the scalability of data processing pipelines, data portability

TABLE 1. A summary of ML use cases, frameworks, data sources, and storage platforms across cases.

Case	Interviewee role	ML use case (type)	Domain*	Data source	Storage	ML framework
A	Chief ML engineer and founder	Object detection (NN)	IoT	Camera	GCP	TensorFlow
B	Chief ML engineer	Form data extraction (NN)	Finance	Internal systems	AWS	TensorFlow
C	Architects (two) and data scientists (two)	Form data extraction (NN)	Public services	Internal systems	On premises	TensorFlow and PyTorch
D	Chief scientific officer	Transcription (NN)	Health care	Internal systems	GCP	Kaldi ASR framework
E	Head of NLU	Speech-based UI (NN)	E-commerce	Open source and end users	GCP	PyTorch
F	ML engineer and founder	ML operations (NN and non-NN)	IT services	Camera	—	Multiple frameworks
G	ML engineer	ML operations (NN and non-NN)	Energy	Internal systems	AWS	TensorFlow and Scikit-Learn
H	Solution architect	Risk management (non-NN)	Finance	Internal systems	AWS	Scikit-Learn and heuristics
I	Data science manager	Predictive maintenance (non-NN)	Engineering	Sensor and technicians	AWS	Spark Analytics and heuristics/rules
J	Data architect	Predictive maintenance (non-NN)	Biochemicals	Sensor	AC	—
K	Data scientist	Anomaly detection (non-NN)	Real estate	Meters	AC	Scikit-Learn and XGBoost
L	Computational biologist	Data analysis (non-NN)	Pharmaceutical	Devices and genomes	AC	R
M	Data scientists (two) and the director of consulting business	Report/document classification (non-NN)	Health care	Internal systems	AC	PyTorch and Scikit-Learn (classification)
N	Principal data scientist	Chatbots and profiling (NN)	Finance	Internal systems	AWS	Watson (IBM) and TensorFlow
O	Solution architects (two)	ML pipeline automation (non-NN)	IT services	Internal systems	AC	—
P	Data scientist	Marketing/campaign management (NN)	Media	Internal systems	AWS	Scikit-Learn, TensorFlow, and fastText

Cases often use cloud storage providers with data centers in Finland or within the European Union, following customer preferences or regulatory constraints.

*of the ML use case

AC: Azure Cloud; ASR: automatic speech recognition; AWS: Amazon Web Services; GCP: Google Cloud Platform; IoT: Internet of Things; NLU: natural language understanding; UI: user interface.

TABLE 2. A summary of practices and challenges.

ML workflow stages	Practices	Challenges
Data management	<ul style="list-style-type: none"> • Batch or stream data loads largely from internal systems, third-party vendors, or devices and sensors • Colocation of data and compute to reduce input–output latency in data transfer • Selecting data storage formats (for example, Apache Parquet) with great consideration of the scalability, portability, and ML frameworks • Data documentation (for example, a data catalog) for fast data identification • Employing data validation approaches (such as descriptive statistics and schema) that are tailored to the types of data • Maintaining data quality by a dedicated team or third-party vendor • Determining data quality metrics from the domain knowledge, especially in highly specialized settings 	<ul style="list-style-type: none"> • Determining the ownership of data quality aspects, especially in large organizations or when data collection is outsourced • IoT-related factors, such as sensor outages, network latency or low traffic priority, sensor quality, and so on • Programming defaults in data collection components, which can lead to poor data quality through subtle, hard-to-notice errors • A lack of standardized annotation formats across DL networks, especially in computer vision, that reduce the interoperability across network architectures
Model training and evaluation	<ul style="list-style-type: none"> • Selecting an ML algorithm based on the available data and learning problem formulation during requirements elicitation and exploratory experiments • Using heuristics to complement the ML algorithm when constrained by regulations or the complexity of models • Employing transfer learning to effectively and accurately train DL models • The flexibility to choose standard ML frameworks, such as TensorFlow and PyTorch (as are popular in DL) as well as Scikit-Learn and XGBoost (in non-DL) • Using ML frameworks that offer great flexibility, efficiency, and usability • Employing multiple approaches to evaluate the quality of ML models, for example, using a validation data set stratified by quality • Managing and tracking model evaluation results using experiment tracking tools or metadata and hash-based approaches 	<ul style="list-style-type: none"> • The cost of training DL models from a clean start, which can be prohibitively high • Determining model explainability • Feature extraction and hyperparameter tuning, which can be time-consuming activities, especially in organization with different types of data • Model benchmarking, which was highlighted as an inherently difficult task, given that it is challenging to replicate publicly available state-of-the-art models and the related results
Model deployment and monitoring	<ul style="list-style-type: none"> • Inference serving through REST-based API endpoints deployed in public cloud environments • Inference serving with strict latency requirements through gRPC endpoints as opposed to REST endpoints • Model deployment for either batch or online inference purposes • Monitoring at different parts of the pipeline to ensure data quality, model quality, and performance and infrastructure utilization 	<ul style="list-style-type: none"> • Deploying models within organizations that do not use the cloud environment, which can be a lengthy process due to relevant data governance protocols • Monitoring the model or data drift in deployed systems, which can be a challenge due to the lack of visibility, especially in scenarios where input data cannot be saved due to GDPR-related constraints
ML pipeline	<ul style="list-style-type: none"> • Version control code and all pipeline-related artifacts (for example, in Git) and a provision execution environment using infrastructure-as-code frameworks (such as Terraform) • Encapsulating ML training workflows in docker containers to increase the portability • Using common container orchestration platforms, such as Kubernetes, to build scalable containerized pipelines • Using ML workflow automation tools, such as Argo and Kubeflow, to execute scheduled ML training pipelines and queues • Tracking ML training experiments largely in custom ways, such as hashing and custom web tools, but also with ML workflow automation tools • Employing CI tools, such as Jenkins, to test and build docker images prior to deployment 	<ul style="list-style-type: none"> • Maintaining an up-to-date stack of tools and frameworks, which requires rigorous testing to avoid regression errors and dependency breaks across toolchains • Pipelines, which can become quite complex, especially when dealing with complex DL architectures where multiple models are maintained • The skills required to run end-to-end automated ML pipelines, which are not easily available

API: application programming interface; DL: deep learning; GDPR: General Data Protection Regulation; gRPC: Google Remote Procedure Call; REST: Representational State Transfer.

between computing environments, and the support of different ML frameworks. Case H uses Apache Parquet instead of the comma- or tab-separated-value file formats commonly used to store structured data for analytics purposes. Case G uses Network Common Data Form to implement a generic data interface to abstract data across ML frameworks and computing platforms.

Data discoverability and accessibility. The discoverability and accessibility of data are emphasized in setups that feature a data lake or where different data types are collected. Case O describes a solution to this problem based on maintaining a data catalog where data and their values are described. Data governance and related processes can limit the use and scope of data accessible for ML purposes.

Data validation. Data validation techniques are commonly applied as a means of controlling data quality. However, data types influence the type of validation approaches used. The validations of image/video, speech, and text tend to require human actors supported by custom tools. For example, a human validator ensures that objects fall within the annotated bounding boxes in an object detection setting. A human speech validator ensures recorded utterances are coherent and consistent with the corresponding text. Case D uses additional heuristics for detecting anomalies between generated texts and submitted utterances. Numerical data types are usually easier to validate automatically.

Data validation in case O is done at the schema and data levels, where dedicated data stewards maintain the schema. Delegating quality control ensures the team managing the data lake ingests data indiscriminately. When data are

sourced from a third-party vendor, the vendor is expected to maintain quality controls (case P).

Data integrity. Data integrity controls ensure data are not changed unexpectedly. Cases D and F apply hashing as part of their data processing pipelines; this ensures training data are verifiable and traceable with respect to a model's lineage. Additionally, this practice ensures that attempts to overwrite data are flagged appropriately. Generally, when hashing is not a suitable approach—for example, when dealing with image files—other custom tooling and heuristics are used to perform anomaly detection; cases B and I make use of this approach.

Data labeling and annotation. The labeling and annotation of data tend to be undertaken manually using custom tools developed to standardize the process. Inconsistent labels are sometimes encountered due to subjective interpretations, resulting in poor data quality. Case B implements a standardized way of normalizing and giving common meaning to concepts to overcome such issues.

Data understanding. This requires domain knowledge for teams to generate valuable insights from data in specialized domains. Domain knowledge is cited as a necessity in the entire lifecycle of the data, for example, handling data from chemical processes or mechanical parts of large systems as represented in cases I, J, and L. In general, challenges in data management practices are mainly attributed to data quality aspects—for example, sensor-related problems, inconsistent labeling, programming errors in data handling software, and so on.

Model training

ML algorithm selection and transfer learning. The choice of ML algorithms is influenced by the training data type and formulation of the learning problem during requirements elicitation. Heuristics are used in cases H and I to complement ML algorithms; in both cases, an explainable decision based on heuristics is preferred compared to an ML solution that has high prediction accuracy but is inexplicable. The ML-heuristic tradeoff tends to arise due to business sector regulatory constraints.

Transfer learning is typically used to train large NNs efficiently, for example, in speech recognition and computer vision settings. This is mainly because model convergence can be a prolonged process that requires significant computing resources. Transfer learning is based on publicly available or proprietary models.

In cases A and F, computer vision systems utilize transfer learning to test different convolutional NN architectures. Case M's natural language processing (NLP) solution is trained using transfer learning to overcome data insufficiency challenges. Case B applies transfer learning based on proprietary models as a cost-management strategy.

Training an NN without transfer learning can be motivated by two factors observed in cases D and E: there are sufficient data and computing resources to train a model to convergence, and there is limited availability of suitable open source models.

ML frameworks. The ML frameworks used across the cases can be broadly categorized as either NN or classical (non-NN) frameworks. TensorFlow-Keras and PyTorch are the two commonly used

frameworks for developing NN models, as summarized in Table 1.

Although ML frameworks may provide similar core features, the choice of the framework can be based on a framework's usability, flexibility, or underlying efficiency in utilizing computing resources. For example, both cases D and E develop automatic speech recognition (ASR) models but use the Kaldi and PyTorch frameworks, respectively. Frameworks can mature into specific domains at varying rates, and, therefore, teams might adopt different frameworks for such historical reasons. Analytics frameworks, such as Spark, also feature in case I. Overall, the challenges in model training relate to the infrastructure costs, complexities of tuning, and identification of explainable factors about a model's performance.

Model evaluation and experiment management

Model training is an iterative process with distinct stages: a determination of the suitability of data and algorithms, parameter and hyperparameter optimization, and model evaluation. Managing metadata from these stages makes the ML workflow traceable and reproducible.

We note three unique approaches used to evaluate models. First, data are stored according to their quality, which enables the composition of data sets with different levels of quality for training and validation purposes (cases D and E). The second approach uses model ensembles, where each model is trained on a unique subset of the data (case B). The third approach applies a configurable inference algorithm where each configuration uses a unique adaptation of the model (case E).

To manage the model evaluation results, case organizations use dedicated experiment tracking tools (cases G, I, N,

O, and P), employ log process metadata (cases B, E, and F) or generate hashes (case D). Hashing involves computing hash values on given combinations of ML artifacts (data, configurations, and models) following the execution of an ML pipeline. Cases E and F generate and collect metadata (such as Git hashes), which are used to produce custom reports. These approaches are summarized in Table 3. The systematic management of experiments facilitates the workflow automation and further increases the traceability and reproducibility of ML workflows.

Model monitoring

Training data drift. This commonly occurs due to structural changes in the data generating process. Identifying drift in numeric data types is achieved by using visual tools, such as graphs or descriptive statistics (cases G, H, and I); image-based data make use of histograms (case F). Speech- and text-based data are also susceptible to drift but can be more challenging to monitor. For example, case D mentioned the recent emergence of the word COVID-19 in the medical sphere, but the word is not available in any historical corpus. Typically, heuristics are used to monitor drift in these speech or NLP settings.

Model drift. This can occur due to data or concept drift, and it is often manifested by a model's gradual or sudden loss of accuracy. Metrics, such as accuracy and error rates, are commonly used to monitor production models. For example, in a transcription setting, the character and word edits required after the inference were measured and used (case D) as error metrics to monitor production models and characterize any model drift.

Infrastructure monitoring. This is applied to ensure models efficiently utilize resources (the GPU/CPU, memory, disk, network, and so on) or flag technical problems, such as scaling designs and I/O bottlenecks during training or inference. Cases D, E, and G closely monitor endpoint latency because it forms an important requirement of the entire ML solution.

TOOLS IN ML PIPELINES (RQ2)

This section presents common tools observed across the cases, as summarized in Table 3.

Version management

The model training code, often written in notebooks, and other project artifacts are version controlled using tools like Git, GitLab, and Bitbucket. Data versioning is done by generating and versioning metadata using specialized tools, such as data version control (DVC). Model training is conducted in public cloud settings for most cases, while a few cases train on the premises. To consistently provision training environments, "infrastructure-as-code" practices and tools, such as Terraform, are used (cases A and E).

ML training workflow

We observe that most case organizations containerize (using docker) individual workflow steps instead of encapsulating all workflow steps in a single container. In ML, containerization facilitates the isolation of different workflow tasks/steps, making the workflow modular, traceable, and reproducible. We further note that containers are commonly orchestrated using Kubernetes, allowing easier migration of pipelines (or parts of them) across infrastructure vendors. Data transfers across workflow steps during training are done using

standard persistent volumes. However, large data sets may require using network mounts (case F).

ML workflows may include steps specifying feature extraction, model training, and validation. The complexity

involved in these steps can vary depending on the ML domain. Workflows can be managed using a custom configuration

TABLE 3. Tools.

Case	Version management	Container platform	ML training workflow	ML experiment tracking	Model repository	ML deployment and serving	Monitoring
A	GitHub, GitLab, and Bitbucket	Kubernetes	Apache Airflow	Tensorboard	GC Container Registry	Embedded with over-the-air updates	Logging and Grafana
B	Git	Kubernetes	Custom	Metadata	—	API endpoint using Bamboo	—
C	GitHub	OpenShift	Custom	None	Nexus and S3 storage	REST API endpoint	Prometheus and Grafana*
D	Git	Kubernetes	Argo	Hashing	—	API	Prometheus and Grafana
E	GitHub	Kubernetes	Custom	Metadata	PostgreSQL	—	Prometheus and Grafana
F	—	Kubernetes	—	Metadata	—	REST API endpoint	Logging, elastic search, and the tool's web UI
G	GitHub	Kubernetes	Custom and Metaflow	MLflow	Docker registry	gRPC API endpoint and Kafka	—
H	Git	AWS Elastic Container	Apache Airflow	—	S3 storage	—	AWS CloudWatch and Splunk
I	GitLab	—	AWS SageMaker	AWS SageMaker	—	—	—
J	—	Kubernetes	Azure ML	—	Azure container registry	Edge server	Azure Monitor
K	Git	—	Azure ML	—	—	Streamlit	—
L	—	—	Azure ML	—	—	R-Shiny apps	—
M	—	—	—	—	Nexus repository	Batch prediction	—
N	Git	AWS Lambda	AWS SageMaker	AWS SageMaker	S3 storage and Databricks model registry*	Batch prediction and Java apps	AWS CloudWatch
O	Git and DVC	Kubernetes	Apache Airflow and Kubeflow	MLflow and Kubeflow	MLFlow model registry	REST API	Logging and Grafana
P	—	—	Databricks	MLflow	S3 storage and MLflow model registry	API, batch, and embedded	—

*Planned—tool information not provided.
Amazon S3: Amazon Simple Storage Service; GC: Google Cloud.

tool [for example, Yet Another Markup Language (YAML) based] or a dedicated workflow toolkit. In complex ML setups, frameworks such as Argo (case D) and Metaflow (case G) are preferred. We note that, although high-level ML workflow platforms, such as AWS SageMaker, provide an end-to-end integration advantage, they are also challenging to use when developing complex models due to inflexibility (cases B and G).

Those in support of custom tooling appreciate the flexibility to add different tools to the workflow. For example, a tool, such as an explainer dashboard, that facilitates a model's explainability is added as part of an ML workflow (case A). An alternative workflow setting can have a single step containing multiple containers (data access data and model training). Customized components that provide access to these containers can be created to monitor the independent utilization of computing resources at the container level (cases C and G).

One overall advantage of using ML workflow tools is that event-based training queues can be orchestrated, for example, based on the continuous arrival of training data. Tools like Apache Airflow provide the functionality to schedule model training based on given triggers.

ML experiments can be tracked using custom web-based user interface tools; this facilitates the evaluation of results and model performance comparison during the development process (cases B and F). To their advantage, custom platforms can freely include any metadata the team consider relevant (case F). Plugins can also be developed to integrate with existing open source solutions, such as MLflow (case G). Low-level training metrics are observed with TensorBoard (case A).

CI and testing

CI tools, such as Jenkins, are used to run tests and build Docker images based on model artifacts resulting from the training workflow (cases A, D, and G). Static code analysis and other tests check general container functionality during the image building process (cases A and G). Domain-specific tests are also executed to ensure the scope of a model's inputs and outputs is unchanged. These tests generally extend testing to the entire pipeline using small amounts of input data (cases D, F, and G).

Docker images created from the CI system are (automatically) deployed to a staging environment for additional tests before deployment to production. Typically, this may include testing the model API's data type (cases A and D), ensuring a model makes sound predictions (case E), and also ensuring that the deployment procedure loads a serialized model into the relevant API endpoints.

Trained models are generally stored in classic data storage solutions or dedicated container image registries. For example, case E stores a model and metadata about the model to a PostgreSQL data warehouse. Case C stores trained models in Nexus, while case G uses a Docker registry to store the models before deployment to production.

ML deployment and serving

Generally, inference serving is done in either a batch or online format. The majority of the models are deployed as Representational State Transfer API endpoints on public cloud or on-premises servers. Other deployment targets include embedding the model on the actual application, such as a mobile application (case P), or deploying it to Internet of Things devices through over-the-air deployments or onsite installations.

Models with strict inference latency requirements are deployed as Google Remote Procedure Call (gRPC) API endpoints. For example, the case C business application has strict latency requirements and, therefore, uses the gRPC, which supports streaming.

Most cases implement a custom serving infrastructure, although emerging model serving systems like KFServing and Seldon are tested in cases C and O, respectively. Finally, we note that data scientists often do not undertake deployment-related tasks, but these are done by other dedicated teams with specialist knowledge, such as the Kubernetes configuration (case G).

Monitoring

After model deployment, monitoring is performed at different levels of granularity. The most common form of monitoring is undertaken for infrastructure management. Logging, monitoring, and alerting tools, like Prometheus and Kubernetes logging (stackdriver), are used to collect a cluster's performance metrics. These metrics are then visualized on dashboards using tools like Grafana, Tableau, or other business analytics tools. For models deployed as APIs, model metrics are logged and stored in Elasticsearch and BigQuery, where the logs are used to perform model health and quality checks in production, such as monitoring average accuracy levels (cases A, F, and O). Overall, maintaining the collective set of tools used across a pipeline can develop into a complex task, especially when dealing with NN architectures.

Because the AI engineering field is still making progress in defining well-established processes, there is a need for details on how such

systems are engineered.¹⁰ Compared to the existing literature, our findings of practices in ML workflows [the “Practices in ML Workflow (RQ1)” section] and tools in ML pipelines [the “Tools in ML Pipelines (RQ2)” section] present the “how” knowledge in contrast to cataloging the practices or tools used in ML applications. This information can be used to objectively analyze practices applied across organizations and identify areas to guide future research efforts that seek to improve knowledge in the field of AI engineering.

Following the taxonomy and challenges described by Lwakatare et al.,² we consider the presented cases to be at the critical deployment stage where ML components need to coexist with other general software components in production systems. We observed that practitioners were implementing practices that also address all of the challenges associated with the critical deployment stage.² In particular, the practices and tools adopted in ML workflows that are summarized in Tables 2 and 3, respectively are indicative of these solutions. However, the challenge related to data management remains an active research area, given the increasing amount of data and disparity in data types. Similarly, the challenges in subsequent stages also need to be addressed, particularly new techniques for monitoring the final models observed in both studies.

Similar to those of Serban et al.,⁵ our results reveal a low adoption of SE best practices in the deployment category and medium to high adoptions in other categories (data, training, team, and coding). We also observed that smaller and younger organizations found it easier to adopt SE practices and emerging tools than older organizations with larger and distributed teams. We attribute

this disparity to legacy systems and processes in the older organizations.

While the survey⁵ indicated that teams with low experience have a low adoption of the SE practices, we observed, on the contrary, that there was a high adoption of the SE practices in teams with limited experience. We attribute this to the field of AI engineering having clearly defined the specific practices (such as the tracking of model experiments) with supporting tools (MLflow and Kubeflow) in academia and industry. There are several areas in the AI engineering field where certain practices, such as data versioning, are considered valid but lack well-established knowledge of how to enact the practice and, thus, are least adopted in the industry.

IMPLICATIONS FOR RESEARCH

Practices and tools for data discoverability

Obtaining good-quality training data for ML purposes is an arduous task—more so due to the increasing amount of data being produced and variability of data handling procedures across data types. Establishing efficient data discoverability procedures would shorten ML production cycles and increase the experimentation of ML models for R&D purposes. Although feature stores have emerged as an intermediate solution for managing data in ML settings, there is a lack of empirical research on their adoption rates, benefits, and applicability across various data types and business settings.

Practices and tools for testing ML models and monitoring them in production

We observed a lack of extensive end-to-end testing of ML pipelines in the

studied cases. In some cases, though, static analyses were performed on the ML training code, and the resulting container images were tested.

We further note that the monitoring of ML-enabled systems needs to extend beyond general infrastructure monitoring practices. A model’s utility can be reduced by degrading accuracy levels as a result of model drift. Although data drift can be detected during model development, concept drift is more challenging to control in production settings. The empirical effects of concept drift and control practices are yet to be explored in the literature.

IMPLICATIONS FOR PRACTICE

Platforms versus independent tools in ML

Generally, we make a similar observation to Xin et al.⁶ that practices in ML workflows and pipelines vary based on factors such as the type of data being used in model training, availability of computing resources, and type of ML solutions being developed. However, we also note two primary ways organizations developed their ML pipelines. First, they can compose a variety of tools to orchestrate a pipeline. Second, teams can use integrated frameworks/platforms, such as SageMaker, which contain built-in tools for various stages of an ML pipeline. Most of the studied teams preferred the first approach because it offers flexibility and the ability to extract low-level information provided by independent tools. However, a common challenge when using separate tools is the required high maintenance efforts. The few teams that use the second approach preferred the

ABOUT THE AUTHORS

DENNIS MUIRURI is a member of the empirical software engineering research group at the University of Helsinki, Helsinki, 00014, Finland. His research interests include the deployment and operations of machine learning systems. Muiruri received an M.Sc. in data science from the University of Helsinki. Contact him at dennis.muiruri@helsinki.fi.

LUCY ELLEN LWAKATARE is a postdoctoral researcher at the University of Helsinki, Helsinki, 00014, Finland. Her research interests include agile, DevOps, and machine learning engineering. Lwakatare received a Ph.D. in software engineering from the University of Oulu. Contact her at lucy.lwakatare@helsinki.fi.

JUKKA K. NURMINEN is a professor at the University of Helsinki, Helsinki, 00014, Finland. His research interests include tools and techniques for data-intensive software systems, the testing of artificial intelligence solutions, and software development for quantum computers. Nurminen received a Ph.D. in systems and operations research from Helsinki University of Technology. He is a Member of IEEE. Contact him at jukka.k.nurminen@helsinki.fi.

TOMMI MIKKONEN is a professor of software engineering at the University of Jyväskylä, Jyväskylä, 40014, Finland. His research interests include the Internet of Things, software architectures, and software engineering for artificial intelligence. Mikkonen received a Ph.D. in software engineering from Tampere University of Technology. Contact him at tommi.j.mikkonen@ju.fi.

instant integration and support offered by platform providers.

Dominant tools

Some well-established tools in SE remain useful when engineering AI systems. Such tools relate to version management (Git), containerization (Docker), and monitoring (Prometheus). However, some of these tools are arguably insufficient for other AI artifacts. For example, code version management tools are not suitable for data version management. Alternative tools dedicated to ML settings are emerging to address some of the inefficiencies encountered by practitioners.

Model inference infrastructure

Following general SE practices, ML API endpoints are based on custom

webservers. However, we note there are initiatives to standardize model serving servers through the open development of a serving API, which is realized by frameworks and tools like NVIDIA's Triton Inference Server, TensorFlow Serving, TorchServe, and so on.

VALIDITY THREATS

Construct validity


Construct validity considers whether the constructs discussed in the interview questions were interpreted in the same way by the researchers and the interviewees. This was mitigated by sharing the study objective and an outline of the interview guide to practitioners before the interview. During the interviews, a brief presentation was given by researchers to communicate

the interview framework, and, later, the discussion was tailored to practitioners' expertise.

External validity

External validity involves the generalization of the findings and other threats that can cause incorrect conclusions to be drawn from the study. Despite having a global presence, the involved organizations are from one geographical location (Finland). This means the conclusions drawn about the state of practice and tools for ML may not be generalizable for the whole SE industry population.

Reliability

Reliability concerns the extent to which the data and analysis are dependent on specific researchers. This threat to validity was mitigated by having at least two researchers throughout the research process. Furthermore, the results were shared with practitioners to review before submission for publication. 

ACKNOWLEDGMENT

This work was supported in part by local authorities (Business Finland) under grant agreements ITEA-2019-18022-IVVES <https://ivves.eu/> ITEA3 program and ITEA-2021-20219-IML4E <https://iml4e.org/> of ITEA4 program.

REFERENCES

1. D. Sculley *et al.*, "Hidden technical debt in machine learning systems," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates Inc., 2015, pp. 2503–2511.
2. L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic, "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *Proc. Agile Processes Softw. Eng. Extreme Program. Conf.*, P. Kruchten, S. Fraser, and F.

- Coallier, Eds. Cham: Springer Nature Switzerland AG, 2019, pp. 227–243.
3. D. Baylor *et al.*, “TFX: A tensor-flow-based production-scale machine learning platform,” in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, ACM, 2017, pp. 1387–1395, doi: 10.1145/3097983.3098021.
 4. K. Hazelwood *et al.*, “Applied machine learning at Facebook: A datacenter infrastructure perspective,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 620–629, doi: 10.1109/HPCA.2018.00059.
 5. A. Serban, K. van der Blom, H. Hoos, and J. Visser, “Adoption and effects of software engineering best practices in machine learning,” in *Proc. 14th ACM Int. Symp. Empirical Softw. Eng. Meas.*, 2020, pp. 1–12, doi: 10.1145/3382494.3410681.
 6. D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, “Production machine learning pipelines: Empirical analysis and optimization opportunities,” in *Proc. 2021 Int. Conf. Manage. Data*, ACM, pp. 2639–2652, doi: 10.1145/3448016.3457566.
 7. S. Amershi *et al.*, “Software engineering for machine learning: A case study,” in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract.*, 2019, pp. 291–300, doi: 10.1109/ICSE-SEIP.2019.00042.
 8. W. Hummer *et al.*, “ModelOps: Cloud-based lifecycle management for reliable and trusted AI,” in *Proc. 2019 IEEE Int. Conf. Cloud Eng.*, pp. 113–120, doi: 10.1109/IC2E.2019.00025.
 9. P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Softw. Eng.*, vol. 14, no. 2, p. 131, 2008, doi: 10.1007/s10664-008-9102-8.
 10. I. Ozkaya, “What is really different in engineering AI-enabled systems?” *IEEE Softw.*, vol. 37, no. 4, pp. 3–6, 2020, doi: 10.1109/MS.2020.2993662.

Over the Rainbow: 21st Century Security & Privacy Podcast

Tune in with security leaders of academia, industry, and government.



Subscribe Today

www.computer.org/over-the-rainbow-podcast