Operationalizing Machine Learning Models - A Systematic Literature Review

Ask Berstad Kolltveit
Norwegian University of Science and Technology
Trondheim, Norway
askbk@stud.ntnu.no

Jingyue Li
Norwegian University of Science and Technology
Trondheim, Norway
jingyue.li@ntnu.no

ABSTRACT

Deploying machine learning (ML) models to production with the same level of rigor and automation as traditional software systems has shown itself to be a non-trivial task, requiring extra care and infrastructure to deal with the additional challenges. Although many studies focus on adapting ML software engineering (SE) approaches and techniques, few studies have summarized the status and challenges of operationalizing ML models. Model operationalization encompasses all steps after model training and evaluation, including packaging the model in a format appropriate for deployment, publishing to a model registry or storage, integrating the model into a broader software system, serving, and monitoring. This study is the first systematic literature review investigating the techniques, tools, and infrastructures to operationalize ML models. After reviewing 24 primary studies, the results show that there are a number of tools for most use cases to operationalize ML models and cloud deployment in particular. The review also revealed several research opportunities, such as dynamic model-switching, continuous model-monitoring, and efficient edge ML deployments.

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → *Software development techniques.*

KEYWORDS

Systematic literature review, Machine learning, MLOps, Deployment, Operationalization

1 INTRODUCTION

Industry adoption of ML is still in its early stages and is rapidly growing. The SE aspect of ML is an active field of research, with the vast majority of work having been done only in the past five years. Existing studies have shown that operationalization of ML is an area that presents practitioners with real challenges [38]. Operationalization, in the context of this paper, consists of taking a trained and evaluated ML model to a serving state in the intended production environment, including necessary support functions, such as monitoring. Tackling operationalization challenges requires adopting good practices and utilizing suitable tooling. Earlier studies have focused mostly on mapping out general SE challenges and practices for ML. This study focuses on 1) researching ML operationalization indepth, and not as part of a broader study of SE for ML and 2) putting more focus on tooling and infrastructure by identifying how current tools are used and what is reported to need further research. The study tries to answer the following research questions.

- **RQ1: How are ML models operationalized in the state of the art?**
- **RQ2: What are the main challenges in operationalizing ML models?**
- **RQ3: What tools and software infrastructure are used to operationalize ML models?**
- **RQ4: What are the feature gaps in the tooling and infrastructure used to operationalize ML models?**

In order to answer the research questions (RQs), a systematic literature review (SLR) was conducted based on the guidelines presented by [25] and [51]. After study selection and quality assessment were performed, 24 primary studies were selected and analyzed. The review results summarise the state-of-the-art techniques and tools for operationalizing ML models. The review also revealed several opportunities for research into better tooling and infrastructure, such as automatic model-switching approaches to meet service level objectives, tools to monitor data and model drift, and efficient solutions for deploying ML models to edge devices.

The rest of this paper is organized as follows. A brief background on ML and DevOps is given in Section 2. Related work is presented in Section 3. The research design and implementation are described in Section 4. The results of the study are presented in Section 5. Section 6 dicusses the research results. Finally, Section 7 concludes the paper.

2 BACKGROUND

ML models have become increasingly prevalent in virtually all fields of business and research in the past decade. [2] reported that 50% of businesses surveyed have adopted artificial intelligence in some business function. With all the research that has been done on the training and evaluation of machine learning models, the difficulty for most companies and practitioners now is not to find new algorithms and optimizations in training, but rather how to actually deploy models to production in order to deliver tangible business value. Most companies are still in the very early stages of incorporating ML into their business processes [46].

2.1 Software Engineering for ML Systems

While traditional SE for non-ML systems is a mature and well-understood field, software engineering for ML systems is still a young and immature knowledge area. Traditional software systems are largely deterministic, computing-driven systems whose behavior is purely code-dependent. ML models have an additional data

, ,

dependency, in the sense that their behavior is learned from data, and they have even been characterized as non-deterministic [18, 36]. The additional data dependency is one of the factors contributing to the fact that ML systems require a great amount of supporting infrastructure.

2.2 DevOps for ML Systems – *MLOps*

DevOps is a subset of software engineering focused on tightening the coupling between the development and operation of software systems. DevOps principles advocate for end-to-end automation [15], which is expressed through the use of version control systems, automated build and deploy pipelines, etc. Some motivating factors for automation are shortening the time to delivery, increasing reproducibility and reducing time spent on automatable processes [14]. DevOps for Machine Learning, named MLOps, is a subset of SE for ML and a superset/extension of DevOps, focusing on adopting DevOps practices when developing and operating ML systems [50]. [13] defines that "ML Ops is a cross-functional, collaborative, continuous process that focuses on operationalizing data science by managing statistical, data science, and machine learning models as reusable, highly available software artefacts, via a repeatable deployment process." [13] identifies four main steps of MLOps: Build, Manage, Deploy and Integrate, and Monitor. Fig. 1 shows the MLOps pipeline proposal in [3].

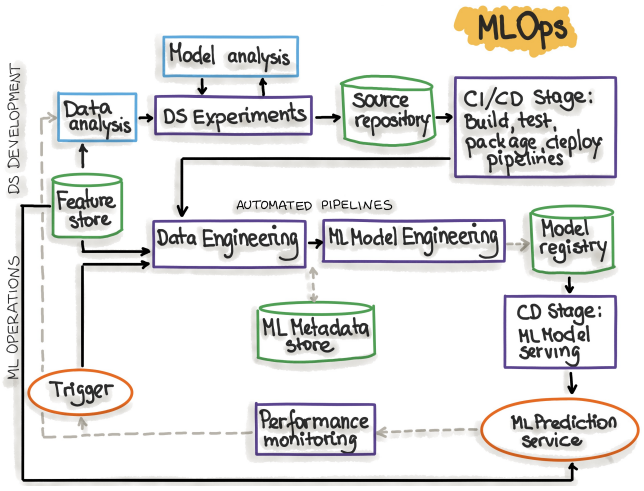

Figure 1: MLOps Pipeline [3]

3 RELATED WORK

Several systematic mapping studies (SMS) and systematic literature reviews (SLRs) have been conducted to summarize SE challenges for ML. [52] was a SMS on the lifecycle of the AI models and found that AI model deployment was underexplored. [36] performed an extensive SMS of SE for AI, investigating reported SE approaches and challenges for developing AI systems. [27] performed an SLR of SE challenges for ML and mapped them to the 12 knowledge areas (KAs) of the SWEBOK [8]. The author found that challenges in SE for ML are found in all KAs, and that safety/security and non-technical areas, in particular, have received a lot of attention in the literature. [37] performed an SLR of challenges and practices in SE for ML. The challenges and practices were mapped to the KAs of the SWEBOK. The authors found that studies from laboratory environments are mainly concerned with building and testing models, which could indicate that more literature produced by industry practitioners should be studied when looking to review the state of the art and state of the practice of deploying models to production. This was further illustrated when the authors highlighted that deployment is one of the areas with the fewest suggested practices in published literature. [35] conducted an SLR on SE for ML using a two-dimensional scheme, categorizing challenges according to quality attributes (adaptability, scalability, privacy and safety) and ML development workflow step (data acquisition, training, evaluation, deployment). The authors also identified solutions to some of the challenges but noted that areas such as privacy and safety, evaluation, and deployment are missing solutions. [18] conducted an SLR of SE for ML in which challenges and solutions were grouped by areas suggested in SWEBOK, similarly to [27], and [37], while including an even greater number of papers than any previous SLR in the field. [31] conducted an SLR of ML model development, identifying phases, techniques, gaps and trends in the ML development lifecycle. The deployment aspect was only briefly mentioned. [41] surveyed the development lifecycle of ML-based IoT application and covered model deployment from declarative ML and deployment, deployment optimization, and Model and action composition perspectives.

Other studies have tried to understand the state of the practice through multivocal literature reviews (MLRs) and interviews or surveys with practitioners. [48] identified 29 best practices of SE for ML by conducting an MLR before conducting a practitioner survey to investigate the adoption and impact of the practices. [23] performed an MLR of the SE lifecycle of ML models, outlining challenges and best practices in each step of the lifecycle. [24] performed an MLR on the adoption of MLOps practices and how companies evolve through different stages of adoption. [24] concludes that successful AI/ML operationalization ensures proper model deployment and updating in different environments. However, the authors of [24] did not study tools and infrastructures for operationalizing ML models. [4] surveyed the academic literature for challenges in deploying ML models. The authors also conducted eleven semi-structured expert interviews with ML practitioners from a wide variety of industries. The authors found that the practitioners' experiences generally confirmed challenges reported in the academic literature. [49] studied software architecture for machine learning, conducting an SLR and practitioner interviews to find architectural challenges and solutions in ML while validating the findings with a practitioner survey. [38] did a survey of case studies on deploying ML models with the goal of outlining a research agenda for addressing the challenges that practitioners were found to encounter. The authors identified practical challenges in 16 different steps of ML deployment, and suggested further research to be done in the areas of tooling and services for individual challenges, as well as new holistic approaches for dealing with ML systems engineering. [33] did an exploratory case study on continuous deployment (CD) for ML. The authors also performed an MLR to create a five-step conceptual process improvement model for the ML deployment

process. The results were validated by conducting a focus group with ten ML practitioners at a telecommunications company.

A common characteristic of existing studies, e.g., [4, 18, 23, 24, 27, 31, 35–38, 41, 48, 49, 52], is that they tend to be quite broad in scope, covering many areas of SE for ML. Only a few of the studies reviewed, e.g., [4, 23, 33, 38, 41], focused on the operationalization aspects of ML models. **However, none of the studies focused on related tooling and infrastructure issues.**

4 RESEARCH DESIGN AND IMPLEMENTATION

The research motivation of this study is to understand the *state-of-the-art in ML deployment.* Differently from those in existing studies, the research questions RQ1 to RQ4 focus on tooling and trying to identify feature gaps reported in studies where model deployment is discussed.

4.1 Primary study search and selection

We followed the SLR guidelines of [25] and [51] with some slight modifications.

Typically, candidate papers in an SLR are identified by constructing a search string for querying digital libraries. However, our string-based searches (using strings such as "machine learning deployment," "MLOps deployment," etc.) in Oria [1], a search engine aggregating research papers from scientific databases, including IEEE Xplore, Springer, ACM Digital library, and Scopus, in late September 2021 failed to produce sufficiently relevant articles on the topic. The articles found did not report deployment details related to the operationalization aspect of SE for ML, with which this study is concerned. This motivated the use of snowballing as an alternative approach for identifying candidate papers. In order to start the snowballing, an initial set of studies was assembled. The studies were mainly found through the reference lists of earlier related work, such as [23] and [36]. Some manual searching with Google Scholar was also done using search strings such as "Mlops deployment" and "machine learning deployment." The study selection criteria found in Table 1 were applied to the start set. The year 2015 was chosen as the earliest year of publication, as this was when research on SE for ML began to gain traction [27], possibly in part due to [47]. From the starting set of 19 papers, 8 satisfied the selection criteria and minimum study quality score, and were thus selected for further snowballing.

The snowballing procedure followed the process proposed by [51], with the exception that forward and backward snowballings were limited to a single iteration each. Backward snowballing from the 8 selected papers provided an additional 21 candidate papers based on skimming of the title/abstract/content of all referred papers from the starting set. Of the 21 candidate papers, 6 papers satisfied the selection criteria and minimum study quality criteria. The quality of included studies was assessed using the criteria found in Table 2, adapted from criteria used by [18] and suggested by [17]. Studies are subject to assessment criteria marked with *X* depending on whether or not they are empirical. For each criterion, the study was awarded 1 point for *yes*, 0.5 for *partly* and 0 for *no*. Sources with an average of 0.5 points or less were excluded. Forward snowballing was conducted using the *Cited by* functionality of Google Scholar. Forward snowballing from the start set yielded 29 initial candidate papers. After selection criteria were applied, 10 candidate papers remained, all of which satisfied the minimum study quality criteria. The total number of papers to review was thus 24.

Table 1: Study selection criteria used in the SLR.

Inclusion criteria	Written in English Published in a peer-reviewed journal, conference or workshop Published after 2015 Discusses one of the following aspects of ML operationalization: challenges, solutions, tooling, processes, requirements Available online
Exclusion criteria	One of the inclusion criteria is not satisfied

4.2 Data extraction and analysis

Data extraction involved reading the literature, extracting information, and entering the data in a spreadsheet. Data synthesis with thematic analysis [30] was performed with the help of TreeSheets[1], a hierarchical spreadsheet application. For each research question, relevant extracted data for each author was entered in the hierarchy. The data pieces were then coded according to their content. Lastly, the hierarchy swap feature was used to reorganize the data from being author-oriented to being code-oriented, which enabled discovery of commonalities and differences between studies.

5 RESEARCH RESULTS

This section presents the results from the SLR by research question.

[1] https://strlen.com/treesheets/

Table 2: Study quality assessment criteria based on criteria from [17] and [18]. Empirical studies are subject to two extra criteria compared with non-empirical studies.

Criteria	Empirical	Non-empirical
Does the author have authority on the subject?	X	X
Does the source have a clearly stated aim?	X	X
Is the author free of any vested interests?	X	X
Have key related sources been linked to or discussed?	X	X
Is relevance (to industry or academia) discussed?	X	X
Does the source have a stated methodology?	X	
Are any threats to validity clearly stated?	X	

5.1 RQ1: How Are ML Models Operationalized in the State-of-the-Art?

5.1.1 Packaging and Integration. Packaging models in containers is a commonly reported method of integration, in which the model is accessible through representation state transfer (REST) or remote procedure call (RPC) interfaces [12, 16, 28, 45]. When deploying numerous different models, using a consistent standard for application programming interfaces (APIs), such as OpenAPI[2], across models can facilitate system integration [16]. Another reported method of integration is to serialize the model (e.g., into ONNX[3], HDF5, Joblib[4]) and load it at runtime, possibly with an ML framework that is optimized for production [9, 21, 39, 40]. The model could also be packaged in a format specific to an end-to-end framework such as MLflow[5] [10]. The model may be integrated directly into the application code [19, 29, 45], traditionally first having to be rewritten for production [21].

5.1.2 Deployment. Deployment, the transitioning of a packaged and integrated model into a serving state, may occur through a few different methods. Models packaged in containers are simply run directly as standalone services [16, 19, 28, 29, 45]. However, models may be deployed in a target environment that is different from where they are packaged, in which case a model transfer must occur, which may happen through either a push- or a pull-pattern.

In a pull-pattern deployment, the target environment (host application running on, e.g., server or edge device) periodically polls for model updates and downloads when available [28, 39, 40].

In a push-pattern deployment, the target environment is notified of the availability of a new model by a master server, e.g., by the server where the model was trained. This will happen either through a messaging service [16, 29], where a message contains metadata including the location of the updated model, or by the model being pushed directly to the target environment through some receiving interface [40].

When an updated model has reached the target environment, subsequent redeployments can happen in a few different ways. For models packaged as standalone containers, a container orchestration service (e.g., Kubernetes[6]) can roll out updated models without downtime. On the other hand, serialized models may simply be loaded into memory by the application, potentially leading to downtime. If the hosting application is containerized, model redeployment may be orchestrated by deploying an entirely new application instance with the new model data alongside the old instance. When the new application instance is finished initializing, it can start serving requests, and the old instance may be evicted [40]. To avoid high response latencies for the first query (cold-start issues), models may be queried with an empty request (or an explicit warmup function call may be used if available) which forces lazy-loaded model components to initialize [28]. When using an ML deployment service (e.g., SageMaker[7]), the service may handle the deployment of serialized models [9]. Models intended for batch predictions may simply be plugged into a computing pipeline such as Apache Spark[8], computing predictions and storing them in a database or data warehouse [28].

5.1.3 Serving and Inference. Models are commonly made available for serving predictions through a REST [12, 16, 26, 29, 40, 45] or RPC [12, 28, 45] API. There are several reported techniques for meeting inference service level objectives (SLOs). One is model-switching, where less accurate but more performant models are used during periods of high load in order to meet required SLOs [54]. Another is using adaptive batching queues with a timeout, so that queries are batched together in batch sizes that are tuned to the individual model and framework. During periods of low traffic, the timeout is reached before the query batch is filled, and inference is run on the batch in order to not exceed the SLO [12]. Additionally, a cache layer may be put on top of the model to reduce computation [12]. In order to avoid cold-start issues caused by model loading and initialization, models should be warmed up at deployment and be kept perpetually warm over the course of their lives [54]. Model warmup can be achieved either through a method provided by the ML framework [28] or more generally through issuing an empty query against the model [16].

5.1.4 Monitoring and Logging. Runtime monitoring is important for operationalized ML in order to increase trust and detect performance degradations [28, 42]. When a model is operational, the whole application stack is constantly monitored for performance metrics such as latency, throughput, disk utilization, etc. [39, 45]. Predictions are logged, and when available joined with actual outcomes [28]. Model accuracy should be used to determine when a new model is needed [39]. The application should be validated against predefined key performance indicators of the project [45].

5.2 RQ2: What Are the Main Challenges in Operationalizing ML Models?

5.2.1 Packaging and Integration. ML frameworks are not all created equal, and some may be better suited for research than production or vice versa, typically because of performance characteristics versus support for rapid experimentation and debugging. [12] noted several challenges in deploying ML frameworks: interface inconsistency across frameworks, changes over time in what is the best framework for the job, and frameworks that are not optimized for deployment. Developers then have to make the choice of either using a suboptimal framework or incurring technical debt in order to support and integrate multiple frameworks. [21] reported solving this tradeoff by decoupling the frameworks from the model by transferring it from research to production using an exchange format for neural networks. [12] also used decoupling as a solution to this problem, but opted for adding an abstraction layer with a simple interface on top of all models. This solution is more general, as it is not specific to neural networks. [7] reported the extensive use of glue code as a challenge with which companies struggle. Glue code is code that merely glues together different parts of the program without itself providing any functionality. The authors also highlight the integration of data-driven models with computation-driven software components as a potentially non-trivial task.

[2]https://www.openapis.org/
[3]https://onnx.ai/
[4]https://joblib.readthedocs.io/en/latest/
[5]https://mlflow.org/
[6]https://kubernetes.io/
[7]https://aws.amazon.com/sagemaker/
[8]https://spark.apache.org/

5.2.2 Serving. Serving-related challenges are some of the ones most often reported in the literature [5–7, 9, 11, 12, 16, 20, 28, 34, 44, 53, 54], usually from a performance perspective.

Achieving a low inference latency and high throughput is widely reported as a challenge in serving ML models [6, 7, 12, 28, 34]. [6] found that there is a statistically significant negative correlation between latency and conversion rate at Booking.com, highlighting the business impact of serving performance. Several potential solutions for improving serving performance have been proposed in the literature. [20] suggested deploying ML models to the edge in order to reduce network latency. However, edge deployment brings a whole host of its own challenges, which will be discussed in Subsubsection 5.2.3. In a system utilizing publish/subscribe messaging services, [44] observed that 33% of application latency is a result of internal communication frameworks, and by experimentation, found that the messaging system becomes congested when faced with an eightfold increase in inference performance. With a growing amount of specialized ML accelerator hardware, it is not inconceivable for such performance bottlenecks to appear in the near future. The authors found that a solution to the problem could be to increase the number of broker node instances used in the messaging system, which clears up the congestion issue. However, ML models typically do not make inferences based on raw data but may require data transformations both before and after inference. [44] warns that the data pre- and post-processing code in streaming ML systems could soon become another performance bottleneck.

Start-up/cold-start latency is a typical serving challenge for ML models [53]. It is mainly caused by the ML model being loaded into memory and initialized, and is typically solved by warming up the model after it has been deployed, and then keeping it perpetually warm [54]. There are, however, problems with this solution, which will be discussed in Subsection 5.4.

[54] discussed the problem of meeting performance SLOs during significant variations in model query traffic, a problem also observed by [53]. The authors noted that currently, the options are to either accept degraded throughput or latency during high-demand scenarios, or to scale up the hardware resources (as suggested by [6]) and thereby incurring increased costs. The difficulty in achieving cost-effective and performant inference was also corroborated by [9], in which the authors observed that increasing hardware instances and network bandwidth both improve inference latencies. The model-switching technique was proposed by [54] to solve the problem of upholding SLOs while minimizing costs. In this technique, high-accuracy/low-performance models are traded for lower-accuracy/high-performance models during load spikes in order to meet the *effective accuracy* SLO, i.e., the average accuracy of predictions that meet latency requirements.

5.2.3 Edge. Edge deployments are often reported as being more challenging than cloud deployments because of hardware resource heterogeneity, the typically distributed nature of edge devices, and possibly varying network connection quality [7, 10, 20, 22, 39, 53]. Our reviewed studies reporting deployment of ML models to Internet of Things (IoT) devices tended to construct custom deployment systems in order to deploy to edge devices [39].

5.2.4 Monitoring and Logging. Monitoring ML models is challenging [6, 28, 41], but is required in order to detect performance degradations and build confidence in the models [7, 10]. [32] observes that "There is a strong desire for continuous monitoring and validation of AI systems post deployment for responsible AI requirements, where current MLOps practices provide limited guidance." After an inference is made, the true label may not be available for an extended period of time, making it difficult to monitor prediction quality [6]. One solution used in practice is to look at the distribution of predictions and determine if there is a significant deviation from what the response distribution of an ideal model would look like [6].

5.3 RQ3: What Tools and Software Infrastructure Are Used to Operationalize ML Models?

In the space of ML-specific platforms, there are a wide array of options reviewed by [45] covering all or parts of the project and model lifecycle. Polyaxon[9], MLflow[10], TFX[11], ZenML[12], Flyte[13], Seldon Core[14] and BentoML[15] are some of the more fully-integrated platforms that exist [10, 45], along with the Amazon-exclusive SageMaker[16] [9, 45]. An overview of various tools for operationalizing ML can be found in [45].

Containerization is a commonly reported method of packaging ML models [39, 42, 45], and Docker[17] is by far the most frequently reported solution [12, 16, 20, 26, 28, 40, 44]. Models packaged in containers are typically accessed through either a REST API [16, 40], such as Flask[18] [20], or RPC interface [12, 45], such as Apache Thrift[19] [28]. Having multiple containers deployed will often necessitate the use of a container orchestration tool such as Kubernetes[20] [29, 40, 43, 44] or Apache Mesos[21] [16]. However, when deploying on edge clusters, K3s[22] may be preferrable [40]. Services running on Kubernetes and K3s may be defined with service descriptors like Helm[23] [40]. In order to manage multiple clusters of containers, a Kubernetes-as-a-service tool such as Rancher[24] may be used [40]. Serverless frameworks like Apache OpenWhisk[25] and Knative[26] may be used to deploy models as functions and provide horizontal scaling [16, 43], both building on top of Kubernetes. When models are downloaded by the target deployment environment, a static storage service such as AWS S3[27] may be used for model storage [9, 29]. A commonly reported messaging and communications

[9] https://polyaxon.com/
[10] https://mlflow.org/
[11] https://www.tensorflow.org/tfx/
[12] https://zenml.io/
[13] https://flyte.org/
[14] https://www.seldon.io/tech/products/core/
[15] https://www.bentoml.ai/
[16] https://aws.amazon.com/sagemaker/
[17] https://www.docker.com/
[18] https://flask.palletsprojects.com/en/2.0.x/
[19] https://thrift.apache.org/
[20] https://kubernetes.io
[21] https://mesos.apache.org/
[22] https://k3s.io/
[23] https://helm.sh/
[24] https://rancher.com/
[25] https://openwhisk.apache.org/
[26] https://knative.dev
[27] https://aws.amazon.com/s3/

framework is Apache Kafka[28], used for logging predictions [28], notifying model hosts of updates [16, 29] and general inter-container communication [44]. A large amount of infrastructure and tooling for ML is Kubernetes-based, reflecting the active ecosystem around the popular container orchestration tool. This could be a barrier to MLOps adoption, as [45] has noted that setting up a Kubernetes cluster is an expensive task.

5.4 RQ4: What Are the Gaps in Current Tooling and Infrastructure Used to Operationalize ML Models?

Many opportunities for further work and research have been identified in the reviewed literature. In particular, [53] and [54] identified a large number of research directions in the area of model serving performance. [7], [42] and [20] have also proposed several areas of research for edge ML deployment.

5.4.1 Model-switching. In the context of an ML system with multiple available variants of the same model (with different accuracy and performance characteristics) and multiple hardware resources to choose from, [53] presents a series of interrelated open problems concerned with serving performance. The automatic selection of model variant based on a given SLO will be referred to as *model-switching* based on the terminology used in [54] and challenges discussed in [41]. The overarching question is that of how the serving system should automatically select a model variant and underlying hardware, given some SLO specified by a client application.

First, given an inference query, how should it be placed/scheduled by the serving system? The options are to either generate a new model variant, load an existing model variant, or query an already loaded model. The choice will depend on the individual application requirements, e.g., whether the inference is online or offline, what the SLOs are, etc. [53] and [54] have proposed that further research be conducted into how to synergistically combine model-switching and hardware resource selection. Further, given the choice of placement, a new model may need to be loaded, additional hardware resources may need to be launched, or loaded models may need to be evicted to free up resources. As [54] has pointed out, keeping all models warm at all times is prohibitively expensive and scales poorly. The model management operations add latency to the query response, affecting the system's ability to meet SLOs.

The next question is how to capture and organize the data needed for making a decision on query placement. In addition to the set of possible hardware resources and model variants, the serving system must also take into account the current state of the system, which is continually changing. The current state of the system is described by which model variants are loaded on which hardware resources, which model variants exist but are not loaded, which model variants do not exist but are able to be generated, the CPU/memory/disk/network usage of each hardware instance, and inference query traffic. All of this data needs to be captured and organized in a way that enables fast and efficient decision making on query placement.

Not only does the logical aspect of hardware and model variant selection need to be considered, the geographical location of hardware resources must also be taken into account. Given that resources may lie anywhere on the core-edge continuum, [53] requested research into the questions of where models should be loaded, where resource management decisions should be made, and where query placement decisions should be made. [43] pointed out that edge resource management from the cloud is complex, partly because nodes may be on private networks or behind firewalls. In addition, edge resources may have unreliable or limited network access, making it challenging to maintain constant communication channels. The authors reported that no out-of-the-box solution currently solves these issues, and request research into transparent edge-cloud resource consolidation/orchestration without the use of point-to-point integrations like VPNs.

[54] further proposed research into combining model-switching with performance optimization techniques like query caching and batching. Several batching techniques have been reported for ML serving systems. [12] used a batching technique with a static maximum batch size with a timeout window which is both tuned on a per-model basis, while [11] proposed an adaptive batching scheme specifically for neural networks, reducing latency during low-traffic scenarios. Combining caching and batching with model-switching is an open problem requiring further work.

[54] proposed research into extending model-switching to additional types of computing resources. CPUs are currently the most widely used for inference in existing ML as a service (MLaaS) platforms. Other types of hardware exist that are suited for neural network inference, such as GPUs, TPUs, etc. Research into model-switching which takes into account these additional types of hardware resources should be conducted to identify possible performance benefits. Further, the additional hardware heterogeneity could pose an interesting challenge when combining model-switching and hardware selection during dynamic query placement.

5.4.2 Edge Deployment. [43] observed that the performance of the Kubernetes scheduler begins degrading with 5,000 nodes and fewer than 10 constraints, struggling to process more than 15 functions per second. It is conceivable that the number of edge devices may reach the thousands and beyond in an IoT scenario, e.g., in industry 4.0, suggesting that the scheduler may become a performance bottleneck in the future. The authors, therefore, proposed research into more scalable function scheduling.

[7] and [42] reported a lack of generic solutions for deploying ML models to embedded and edge devices. [42] reported that conventional IoT provisioning frameworks are not suited for deploying ML models, while [20] reported that there is a lack of solutions for deploying to edge devices that do not support containerization techniques. [43] reported that support for non-x86 architectures is lacking.

Edge devices present greater heterogeneity in computing capabilities, storage and networking. In the context of having multiple models deployed in an edge architecture, some models may have to be evicted from time to time in order to free up resources for new deployments. [42] and [43] proposed research into strategies for intelligently evicting models from edge devices. According to [42], monitoring edge-deployed ML models can potentially present two different challenges. First, some metrics, such as concept drift and model drift, require continuous access to the original training

[28] https://kafka.apache.org/

set. However, edge devices may not have access to the training set for multiple reasons, including storage space constraints and data privacy concerns. Secondly, [42] identified the problem of triggering centralized model retraining. Retraining triggers require that monitoring data be continuously streamed from the edge devices to the (possibly centralized) ML pipeline. Not only could this be difficult because of unreliable network connections, bandwidth restrictions or data usage constraints, handling large volumes of incoming monitoring data from edge devices may itself present a challenge of scalability. On the topic of monitoring, [10] highlighted a need for framework-agnostic model telemetry solutions to enable the capture of statistical performance metrics with a broad set of supported deployment environments. [10] also noted the need for a general representation format for multistep ML workflows. Many processes in ML pipelines contain multiple subtasks, where changing a single step may cause regressions. A new format should, according to the authors, provide explicit IO interfaces and enable parallel execution of independent subtasks with existing workflow execution systems (e.g., Airflow[29]).

6 DISCUSSION

Compared with related work, this study focused more on the tooling and infrastructure aspects of MLOps. The results of the review highlight two areas where further research should be pursued. The first major area is performance and scalability in prediction serving, possibly using a combination of existing techniques for batching, caching and dynamic model selection. The second major area with a reported need for further research is deployment and monitoring of models in edge environments.

The results of the study could be of value to industry practitioners as a reference for various techniques and patterns for operationalizing ML. In addition, the study could also serve as a starting point for understanding what tools are used in different stages and contexts of operationalization, as well as what challenges may be expected when operationalizing ML models.

There are certain inherent threats to validity in an SLR, such as the possibility that not all relevant literature is found or included. To mitigate this, the starting set included papers spanning multiple years (2017-2021), venues (IEEE, ACM, Sciendo, Frontiers, Sciencedirect, Zenodo, IGI Global, Wiley, MDPI, Springer) and authors. However, as the snowballing procedure was only limited to a single iteration of forward and backward snowballing, instead of continuing until no new studies are found, it is quite probable that this review is not exhaustive. Furthermore, qualitative data analysis is closely tied to the researchers' background, identity, assumptions and beliefs. To address this issue, the data analysis results were discussed between the authors to arrive at a consensus when needed. The initial string-based search possibly failed because too many aspects were attempted covered in just a few keywords, instead of using a more specific search string (such as "machine learning AND (serving OR deployment OR packaging)").

7 CONCLUSION AND FUTURE WORK

After reviewing 24 primary studies and focusing on understanding the state of the art and the challenges of operationalizing ML models, our study has contributed the following new knowledge: 1) How ML models are operationalized with respect to tools and infrastructure. This includes various techniques for packaging and integration, deployment, serving and monitoring, which, to the best of the authors' knowledge, have not been investigated in any previous SLR. 2) An overview of what tools have reportedly been used to operationalize ML models in the literature. 3) An overview of areas for further research into ML operationalization as suggested by the literature.

The study potentially misses out on practitioner knowledge. A natural next step would thus be to conduct a GLR to investigate practitioner knowledge on the subject.

[29]https://airflow.apache.org/

REFERENCES

[1] 2020. Oria Search Engine. http://oria.no/ Accessed 05.09.2020.

[2] 2020. The state of AI in 2020. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/global-survey-the-state-of-ai-in-2020

[3] 2022. MLOps pipeline. https://ml-ops.org/content/mlops-principles.

[4] Lucas Baier, Fabian Jöhren, and Stefan Seebacher. 2019. Challenges in the deployment and operation of machine learning in practice. In *ECIS 2019 - 27th European Conference on Information Systems*. https://www.researchgate.net/publication/332996647_CHALLENGES_IN_THE_DEPLOYMENT_AND_OPERATION_OF_MACHINE_LEARNING_IN_PRACTICE

[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX. ACM. https://doi.org/10.1145/3097983.3098021

[6] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 2019. 150 Successful Machine Learning Models. ACM. https://doi.org/10.1145/3292500.3330744

[7] Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2021. Engineering AI Systems. IGI Global, 1–19. https://doi.org/10.4018/978-1-7998-5101-1.ch001

[8] Pierre Bourque. 2014. *SWEBOK : guide to the software engineering body of knowledge*. IEEE Computer Society, Los Alamitos, CA.

[9] Dheeraj Chahal, Ravi Ojha, Sharod Roy Choudhury, and Manoj Nambiar. 2020. Migrating a Recommendation System to Cloud Using ML Workflow. ACM. https://doi.org/10.1145/3375555.3384423

[10] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. 2020. Developments in MLflow. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*. ACM. https://doi.org/10.1145/3399579.3399867

[11] Yujeong Choi, Yunseong Kim, and Minsoo Rhu. 2021. Lazy Batching: An SLA-aware Batching System for Cloud Machine Learning Inference. IEEE. https://doi.org/10.1109/hpca51647.2021.00049

[12] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[13] Sweenor David, Hillion Steven, Rope Dan, Kannabiran Dev, Hill Thomas, O'Connell Michael, and Safari an O'Reilly Media Company. 2020. ML Ops : Operationalizing Data Science. https://go.oreilly.com/queensland-university-of-technology/library/view/-/9781492074663/?ar

[14] Breno B. Nicolau de França, Helvio Jeronimo, and Guilherme Horta Travassos. 2016. Characterizing DevOps by Hearing Multiple Voices. ACM Press. https://doi.org/10.1145/2973839.2973845

[15] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. DevOps. 33, 3 (may 2016), 94–100. https://doi.org/10.1109/ms.2016.68

[16] Alvaro Lopez Garcia, Jesus Marco De Lucas, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germen Molto, Marcin Plociennik, Viet Tran, Andy S. Alic, Miguel Caballer, Isabel Campos Plasencia, Alessandro Costantini, Stefan Dlugolinsky, Doina Cristina Duma, Giacinto Donvito, Jorge Gomes, Ignacio Heredia Cacha, Keiichi Ito, Valentin Y. Kozlov, Giang Nguyen, Pablo Orviz Fernandez, Zdenek Sustr, and Pawel Wolniewicz. 2020. A Cloud-Based Framework for Machine Learning Workloads and Applications. 8 (2020), 18681–18692. https://doi.org/10.1109/access.2020.2964386

[17] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2016. The need for multivocal literature reviews in software engineering. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*.

ACM. https://doi.org/10.1145/2915970.2916008
[18] Görkem Giray. 2021. A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software* 180 (oct 2021), 111031. https://doi.org/10.1016/j.jss.2021.111031
[19] Tuomas Granlund, Vlad Stirbu, and Tommi Mikkonen. 2021. Towards Regulatory-Compliant MLOps: Oravizio's Journey from a Machine Learning Experiment to a Deployed Certified Medical Product. 2, 5 (jun 2021). https://doi.org/10.1007/s42979-021-00726-1
[20] Nitu Gupta, Katpagavalli Anantharaj, and Karthikeyan Subramani. 2020. Containerized Architecture for Edge Computing in Smart Home : A consistent architecture for model deployment. IEEE. https://doi.org/10.1109/icci48352.2020.9104073
[21] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. IEEE. https://doi.org/10.1109/hpca.2018.00059
[22] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, Kaoutar El Maghraoui, Anupama Murthi, and Punleuk Oum. 2019. ModelOps: Cloud-Based Lifecycle Management for Reliable and Trusted AI. IEEE. https://doi.org/10.1109/ic2e.2019.00025
[23] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. 2021. Architecting AI Deployment: A Systematic Review of State-of-the-Art and State-of-Practice Literature. In *Lecture Notes in Business Information Processing*. Springer International Publishing, 14–29. https://doi.org/10.1007/978-3-030-67292-8_2
[24] Meenu Mary John, Helena Holmstrom Olsson, and Jan Bosch. 2021. Towards MLOps: A Framework and Maturity Model. IEEE. https://doi.org/10.1109/seaa53835.2021.00050
[25] B. Kitchenham and S Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering.
[26] Rajalakshmi Krishnamurthi, Raghav Maheshwari, and Rishabh Gulati. 2019. Deploying Deep Learning Models via IOT Deployment Tools. IEEE. https://doi.org/10.1109/ic3.2019.8844946
[27] Fumihiro Kumeno. 2020. Sofware engneering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13 (2020), 463–476. https://doi.org/10.3233/IDT-190160
[28] Li Erran Li, Eric Chen, Jeremy Hermann, Pusheng Zhang, and Luming Wang. 2017. Scaling Machine Learning as a Service. In *Proceedings of The 3rd International Conference on Predictive Applications and APIs (Proceedings of Machine Learning Research, Vol. 67)*, Claire Hardgrove, Louis Dorard, Keiran Thompson, and Florian Douetteau (Eds.). PMLR, 14–29. https://proceedings.mlr.press/v67/li17a.html
[29] Yan Liu, Zhijing Ling, Boyu Huo, Boqian Wang, Tianen Chen, and Esma Mouine. 2020. Building A Platform for Machine Learning Operations from Open Source Frameworks. 53, 5 (2020), 704–709. https://doi.org/10.1016/j.ifacol.2021.04.161
[30] Chad Lochmiller. 2021. Conducting Thematic Analysis with Qualitative Data. *The Qualitative Report* (jun 2021). https://doi.org/10.46743/2160-3715/2021.5008
[31] Giuliano Lorenzoni, Paulo Alencar, Nathalia Nascimento, and Donald Cowan. 2021. Machine Learning Model Development from a Software Engineering Perspective: A Systematic Literature Review. (Feb. 2021). arXiv:2102.07574 [cs.SE]
[32] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, David Douglas, and Conrad Sanderson. 2021. Software Engineering for Responsible AI: An Empirical Study and Operationalised Patterns. https://doi.org/abs/2111.09478
[33] Lucy Ellen Lwakatare, Ivica Crnkovic, Ellinor Rånge, and Jan Bosch. 2020. From a Data Science Driven Process to a Continuous Delivery Process for Machine Learning Systems. In *Product-Focused Software Process Improvement*. Springer International Publishing, 185–201. https://doi.org/10.1007/978-3-030-64148-1_12
[34] Lucy Ellen Lwakatare, Aiswarya Raj, Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2019. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. Springer International Publishing, 227–243. https://doi.org/10.1007/978-3-030-19034-7_14
[35] Lucy Ellen Lwakatare, Aiswarya Raj, Ivica Crnkovic, Jan Bosch, and Helena Holmström Olsson. 2020. Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and Software Technology* 127 (nov 2020), 106368. https://doi.org/10.1016/j.infsof.2020.106368
[36] Silverio Martínez-Fernández, Xavier Franch, Andreas Jedlitschka, Marc Oriol, and Adam Trendowicz. 2021. Developing and Operating Artificial Intelligence Models in Trustworthy Autonomous Systems. Springer International Publishing, 221–229. https://doi.org/10.1007/978-3-030-75018-3_14
[37] Elizamary Nascimento, Anh Nguyen-Duc, Ingrid Sundbø, and Tayana Conte. 2020. Software engineering for artificial intelligence and machine learning software: A systematic literature review. (Nov. 2020). arXiv:2011.03751 [cs.SE]
[38] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2020. Challenges in Deploying Machine Learning: a Survey of Case Studies. *The ML-Retrospectives, Surveys & Meta-Analyses Workshop, NeurIPS 2020*, Article arXiv:2011.09926 (Nov. 2020). arXiv:2011.09926 [cs.LG] https://ui.adsabs.harvard.edu/abs/2020arXiv201109926P
[39] Devon Peticolas, Russell Kirmayer, and Deepak Turaga. 2019. Mímir: Building and Deploying an ML Framework for Industrial IoT. IEEE. https://doi.org/10.1109/icdmw.2019.00065
[40] Pekka Pääkkönen, Daniel Pakkala, Jussi Kiljander, and Roope Sarala. 2020. Architecture for Enabling Edge Inference via Model Transfer from Cloud Domain in a Kubernetes Environment. 13, 1 (dec 2020), 5. https://doi.org/10.3390/fi13010005
[41] Bin Qian, Jie Su, Zhenyu Wen, Devki Nandan Jha, Yinhao Li, Yu Guan, Deepak Puthal, Philip James, Renyu Yang, Albert Y. Zomaya, Omer Rana, Lizhe Wang, Maciej Koutny, and Rajiv Ranjan. 2020. Orchestrating the Development Lifecycle of Machine Learning-Based IoT Applications: A Taxonomy and Survey. *ACM Comput. Surv.* 53, 4, Article 82 (aug 2020), 47 pages. https://doi.org/10.1145/3398020
[42] Thomas Rausch and Schahram Dustdar. 2019. Edge Intelligence: The Convergence of Humans, Things, and AI. IEEE. https://doi.org/10.1109/ic2e.2019.00022
[43] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. 2019. Towards a Serverless Platform for Edge AI. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotedge19/presentation/rausch
[44] Daniel Richins, Dharmisha Doshi, Matthew Blackmore, Aswathy Thulaseedharan Nair, Neha Pathapati, Ankit Patel, Brainard Daguman, Daniel Dobrijalowski, Ramesh Illikkal, Kevin Long, David Zimmerman, and Vijay Janapa Reddi. 2020. Missing the Forest for the Trees: End-to-End AI Application Performance in Edge Data Centers. IEEE. https://doi.org/10.1109/hpca47549.2020.00049
[45] Philipp Ruf, Manav Madan, Christoph Reich, and Djaffar Ould-Abdeslam. 2021. Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. 11, 19 (sep 2021), 8861. https://doi.org/10.3390/app11198861
[46] Stephan Schlögl, Claudia Postulka, Reinhard Bernsteiner, and Christian Ploder. 2019. Artificial Intelligence Tool Penetration in Business: Adoption, Challenges and Fears. Springer International Publishing, 259–270. https://doi.org/10.1007/978-3-030-21451-7_22
[47] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28.
[48] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. 2020. Adoption and Effects of Software Engineering Best Practices in Machine Learning. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM. https://doi.org/10.1145/3382494.3410681
[49] Alex Serban and Joost Visser. 2021. An Empirical Study of Software Architecture for Machine Learning. (May 2021). arXiv:2105.12422 [cs.SE]
[50] Julian Soh and Priyanshi Singh. 2020. Machine Learning Operations. In *Data Science Solutions on Azure*. Apress, 259–279. https://doi.org/10.1007/978-1-4842-6405-8_8
[51] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. ACM Press. https://doi.org/10.1145/2601248.2601268
[52] Y. Xie, L. Cruz, P. Heck, and J. S. Rellermeyer. 2021. Systematic Mapping Study on the Machine Learning Lifecycle. In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. IEEE Computer Society, Los Alamitos, CA, USA, 70–73. https://doi.org/10.1109/WAIN52551.2021.00017
[53] Neeraja J. Yadwadkar, Francisco Romero, Qian Li, and Christos Kozyrakis. 2019. A Case for Managed and Model-less Inference Serving. ACM. https://doi.org/10.1145/3317550.3321443
[54] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association. https://www.usenix.org/conference/hotcloud20/presentation/zhang