Architectural Design Decisions for Machine Learning Deployment

I. INTRODUCTION

Software engineering (SE), software architecture (SA), and machine learning (ML) are all established disciplines, but a significant disparity between SE/SA on the one hand and ML on the other has emerged, even though ML often involves software engineering activities such as development, maintenance and deployment [1]–[3]. ML practitioners have various architectural design decision (ADD) options from which to choose. Since SE/SA approaches to ML are still at a relatively early stage compared to traditional SE/SA practices, in addition to the lack of systematic approaches to solution building and systems design, it may be hard for practitioners to know which options are available to choose from, especially when to decide on specific options.

We conducted a qualitative Grey Literature Study (GLS) [4], [5], based on Grounded Theory (GT) [6]–[9], to formalise current practitioner understanding and architectural concepts of ML solution deployment. Other detailed aspects of ML engineering such as data processing, big data and development environments did not specifically form part of this study. We covered data processing similarly in a prior publication [10] and may address the remaining topics in future papers. We set out to address the following research questions:

- **RQ1** Which architectural design decisions are available to choose from in the context of deployment for ML, and what are their corresponding decision options?
- **RQ2** What are the relations between these decisions and their decision options?
- **RQ3** Which decision drivers (forces) are relevant to the decision options?

The result was a formal, UML-based model of ADDs, decision options, decision drivers, and their relations in the domain of deployment for ML that serves to guide practitioners and further the scientific knowledge of their practices, concerns and understanding in the field of deployment for ML.

We discuss related studies in the field in Section II and describe our research method in Section III. After that, in Section IV we present the various ADDs, decision options, decision drivers and their relations – these represent the results of our empirical study. Finally, in Section V we interpret our results and consider possible threats to validity before concluding in Section VI.

II. RELATED WORK

There appear to be relatively few standardised methods and processes in SE for ML. Thus, ML engineers wishing to apply professional SE techniques have to contend with a multitude of engineering and architectural decisions to carry out their activities.

Washizaki et al. [11] conducted a systematic literature review, which categorised and detailed a collection of software (anti-) patterns for ML. Like our study, their work is based on a GLS, but it also includes scientific literature sources. In contrast to ours, their work focuses on ML pipeline and software development patterns, not deployment patterns.

Sculley et al. [1] describe the technical debt incurred when maintaining production ML systems, such as entanglement, data dependencies, configuration challenges, reproducibility debt and system anti-patterns. Some of these factors are important motivations for our work and, in part, appear in our decision drivers. In contrast to our work, they do not provide detailed ADDs for ML deployment.

Lwakatare et al. [2] carried out an empirical investigation into software engineering challenges for ML systems, devising a taxonomy of common issues, including data dependency management, deployment difficulties and result reproduction.
challenges, without formally modelling design decisions.

Bosch et al. [12] document an overview of the software engineering challenges associated with ML solutions, such as model management, deployment, data pipeline challenges, monitoring, logging, design methods and data quality management. They also identify unresolved research areas, including distributed model creation, distributed data storage, data generation and automated experimentation. Whereas our work concentrates on concrete ADDs for deployment, their work focuses on identifying challenges (which, in part, appear in our decision drivers) and categorising relevant practices at a high level of abstraction.

Nascimento et al. [3] provide a comprehensive systematic literature review on software engineering for ML, document the state of the art and identify open challenges, mainly in testing, AI software quality, and data management. They found that most SE practices proposed are guidelines, lessons learned, and tools. We propose a formalised guidance model in the form of ADDs for the deployment of ML models, a field not yet adequately covered in the literature.

Mäkinen et al. [13] studied the importance of MLOps to practitioners for rapid deployment in ML via a survey. They discovered that most respondents were focused on how best to use data, the initial building and deployment of models and that MLOps is only beneficial when retraining and redeploying models. Unlike our approach, they did not develop a model of ADDs in the domain of ML deployment. Despite their findings and the overlap with the subject of this study, our research indicated that MLOps is a complex topic and includes many non-deployment-related ADDs. It was thus deemed out-of-scope.

Our approach was to reduce the gap between science and practice by studying methods and techniques documented by ML practitioners in the context of deployment. We formally modelled ADDs, decision options, practices, decision drivers, and their relations. Valuable insights were gained, which may guide practitioners in selecting suitable solutions to recurring design decisions, help practitioners overcome challenges, mitigate problems and avoid suboptimal decisions. To keep this publication concise, we restrict ourselves to decisions associated with the deployment of ML models. To our knowledge, this is one of the first studies of its kind in the field.

III. RESEARCH METHOD

We conducted a GLS as a systematic investigation into practitioner understanding of the deployment of ML models using practitioner sources exclusively. Applying GT, we coded the encountered phenomena in our sources and developed a formal theory encoded as a UML-based model. We successfully applied GT combined with a GLS in our previous studies [10], [15], [16] and describe the method with the reasoning for its application below.

1All project artefacts, including those for MLOps, are provided in our replication package [14].

A. Grounded Theory

GT is a qualitative, inductive research method that links data analysis with theory. Iterative steps are taken when interpreting data, whereby the focus and central goal is to build a theory grounded in said data. Data analysis should occur during data collection and not afterwards.

The most important activity in GT is constant comparison: the researcher continuously and iteratively compares pre-existing data and concepts with new data. Any newly-arising abstract concepts should then be compared with pre-existing concepts and data. The concepts are organised into categories (or codes) and are compared and linked to properties and each other via relations [9]. The concepts, categories and properties derived from the data should guide the next iteration of research activities.

Another central activity is memo-writing, which documents the theory-building process, provides a basis for reflection, improves transparency by creating an audit trail, and facilitates reproducibility. Theoretical sampling involves actively seeking out new data based on the results of the previous iteration, considering the kinds of data that should be collected next [17]. This is continued until theoretical saturation is reached, i.e. “the point in category development at which no new properties, dimensions, or relationships emerge during analysis” [7].

We applied the methodology of Strauss and Corbin [9], which is characterised by three types of coding activity:

- Open coding involves developing concepts based on the data sources. It entails asking specific (and consistent) questions of the data, precise (and consistent) coding, and memo writing with minimal assumptions.
- Axial coding is the development of categories and the linking of data, concepts, categories and properties.
- Selective coding refers to the integration of the categories that have been developed and their grouping around a central core category.

B. Grey Literature

According to Garousi et al. [5], grey literature in the context of software engineering is “any material about SE that is not formally peer-reviewed nor formally published”, such as blog posts, articles, presentations and audio-video material [4]. Grey literature was chosen as the sole data source for this study because we wished to focus on understanding practitioners’ views within this domain, and such data sources are most representative of these views. This is confirmed by Rainer and Williams [18], who describe various benefits to grey literature sources in software engineering research, including that they “promote the voice of the practitioner” and “provide information on practitioners’ contemporary perspectives on important topics relevant to practice and to research”.

C. Methodology

Figure 1 illustrates our research method. We searched for practitioner sources using standard search engines (e.g., Google, StartPage, DuckDuckGo) and topic portals (e.g., InfoQ, DZone). The search term for the initial source was
“machine learning deployment”. We then repeatedly and iteratively applied GT coding practices and constant comparison to identify concepts, categories, properties, and relations. The resulting entities were modelled in Python code, from which a UML model was generated. The subsequent sources were then searched for using appropriate search terms based on topics identified in the previous iteration and guided by the research so far, with particular consideration given to topics needing coding and their potential contribution to the model. Practitioner articles were deemed candidates if they were relevant to the topic under consideration and did not appear to be marketing a business or product, and each author reviewed the other author’s selection of sources for suitability.

![Fig. 1. Research Method](image)

We applied open coding to transform conceptual details into conceptual labelling. Next, while axial coding, we identified categories based on recurring, synonymous and related concepts. During open and axial coding, we studied each source line by line. Our thought processes, conceptual understanding, interpretation and reasoning behind each coding decision was documented in memos for the various sources, facilitating traceability of codes back to their sources. While selective coding, we carved out the main ideas of the theory, gaining an understanding of the big picture by considering the data and analysis results. We also revisited and refined previous sources during selective coding. Finally, we used formal UML-based modelling for axial and selective coding to develop a precise, consistent theory as a formal UML model. Once around five to seven additional sources no longer added value to our model, theoretical saturation was understood to have been reached. Our knowledge sources are summarised in Table I.

Modelling was achieved using CodeableModels\(^2\) – a Python tool that can be used to define models and model instances in code and record memos. Based on the Python code, PlantUML\(^3\) code generators were used to generate graphical visualisations of the model\(^4\), in addition to a textual model specification in Markdown\(^5\) and Latex\(^6\).

### IV. Architectural Design Decisions

This section presents our study results as ADDs derived from practitioner views and practices sourced from grey literature within the context of deploying ML models. Table II presents an overview of the various ADDs, decision options, grey literature sources\(^6,7\) and decision drivers (forces)\(^8\), the associated ADDs for which are described in this section.

#### A. Deployment Approach Decision

Deployment in an ML context is the process of making a trained ML model available so clients may avail of its predictions. When planning to deploy ML models, a decision surrounding the level of automation that shall be applied must be made. Our sources identified three initial decision options addressing automated deployments, each encompassing varying levels of automation.

As illustrated in Figure 2, the trivial decision solution is to opt for no automated deployment; however, this option hinders iterative development and precludes process and work automation, along with reducing observability. An alternative deployment decision option that offers at least some degree of automated delivery is building, testing, and deploying models to be served and other system components in a semi-automated fashion by deploying pre-prepared pipelines. This process is considered semi-automated because even though the pipeline execution itself can be automated, the pipeline itself must be manually created each time it changes in some aspect. The advantage of adopting this approach is increased support for process and work automation and iterative development, plus increased observability.

An option that yields more automation is to build, test, and deploy machine learning models based on CI/CD pipeline automation. “CI/CD” stands for “continuous integration/continuous delivery”. CI is “the practice of building and running automated tests against every change you make to your application so you can ensure that your software is always in a working state.”\(^20\), whereas CD “provides the ability to release new, working versions of your software several times a day”\(^20\) and thus CI/CD is understood as the combination of both approaches. Applying this level of automation leads to a dramatic increase in support for all three forces. Noteworthy is that both the semi-automated and CI/CD pipeline decision options can both include a data pipeline (automation of the data processing steps involved in building an ML model), a model building pipeline (automation of the training and evaluation steps of building an ML model), or both, since these practices may also be combined.

A related decision to consider, either from the outset or when evaluating whether to apply CI/CD pipelines, is a more inclusive approach, specifically how to automate integration and delivery in a machine learning context. We shall consider this decision in Section IV-C. Finally, should MLOps be applied and, if so, when is another related decision and is discussed in Section IV-B.

\(^1\)The following colour scheme in Table II indicates the source and evidence frequency:
- $\square < 5\%$
- $\square < 10\%$
- $\square < 20\%$
- $\square < 35\%$
- $\square < 50\%$
- $\square < 70\%$
- $\geq 70\%$

\(^2\)Source archive URLs may be found in our replication package \[14\].

\(^3\)Force impacts in Table II range from “very negative” (−) to “very positive” (+) based on our interpretation of the grey literature sources.

---

2\(http://github.com/uzdun/CodeableModels\)
3\(http://plantuml.com\)
4\(\text{Generated UML figures have been optimised for space and readability.}\)
5\(https://daringfireball.net/projects/markdown\)
<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Source Type</th>
<th>Example</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>How to power up your product by machine learning with python microservice, pt. 1</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s2</td>
<td>Architecting a Machine Learning Pipeline: How to build scalable Machine Learning systems - Part 2/2</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s3</td>
<td>Some Thoughts on Modularization in Machine Learning</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s4</td>
<td>Productionizing Machine Learning with a Microservices Architecture</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s5</td>
<td>Microservices Suck for Machine Learning (and what we did about it)</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s6</td>
<td>MLOps: Continuous delivery and automation pipelines in machine learning</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s7</td>
<td>Composing Deep-Learning Microservices for the Hybrid Internet of Things</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s8</td>
<td>Continuous Intelligence: Moving Machine Learning Application into Production Reliably</td>
<td>Slides</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s9</td>
<td>Architecture of a real-world Machine Learning system</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s10</td>
<td>Architecting a Machine Learning System for Risk</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s11</td>
<td>Architecting a Scalable Real Time Learning System</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s12</td>
<td>System Architectures for Personalization and Recommendation</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s13</td>
<td>Architectural thinking in the Wild West of data science</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s14</td>
<td>Machine Learning Architecture: The Core Components</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>s15</td>
<td>Scalable Software and Big Data Architecture - Big Data and Analytics Architectural Patterns</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s16</td>
<td>Machine Learning in Production: Software Architecture</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s17</td>
<td>AutoML</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s18</td>
<td>AutoML is Overhyped</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s19</td>
<td>Three Levels of ML Software</td>
<td>Practitioner Audience Article</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s20</td>
<td>MLOps: Methods and Tools of DevOps for Machine Learning</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s21</td>
<td>MLOps: What It Is, Why it Matters, and How To Implement It (from a Data Scientist Perspective)</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s22</td>
<td>MLOps Principles</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s23</td>
<td>Machine Learning Monitoring: What It Is, and What We Are Missing</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s24</td>
<td>Automated monitoring of your machine learning models with Amazon SageMaker Model Monitor and sending predictions to human review workflows using Amazon A2I</td>
<td>Blog Post</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s25</td>
<td>MLOps: Model management, deployment, and monitoring with Azure Machine Learning</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s26</td>
<td>The Pros and Cons of Using Jupyter Notebooks as Your Editor for Data Science Work TL;DR: PyCharm’s probably better</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s27</td>
<td>10 reasons why data scientists love Jupyter notebooks</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s28</td>
<td>5 reasons why jupyter notebooks suck</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s29</td>
<td>Jupyter Notebook is the Cancer of ML Engineering</td>
<td>Practitioner Audience Article</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s30</td>
<td>Comparing Data Version Control Tools - 2020</td>
<td>Blog Post</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s31</td>
<td>How to test your ML models from hypothesis to production</td>
<td>Blog Post</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>s32</td>
<td>Evaluate ML Classifier Performance using Statistical Hypothesis Testing in Python</td>
<td>Blog Post</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s33</td>
<td>Testing Your Machine Learning Pipelines</td>
<td>Blog Post</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s34</td>
<td>Performance Testing ML Serving APIs With Locust</td>
<td>Blog Post</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>s35</td>
<td>Machine Learning at Scale with Parallel Processing</td>
<td>Blog Post</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

![Fig. 2. Deployment Approach Decision](image-url)
B. MLOps Decision

Should MLOps be applied and, if so, when? is a recurring decision throughout this study. MLOps has become increasingly relevant to practitioners [13] and encompasses techniques for the rapid deployment of ML models and includes data-related practices such as data sourcing, data analysis and data labelling; model-related practices like model benchmarking, model training and model validation; and various other practices including experiment tracking, hardware scaling and model service profiling. As evident in Table II MLOps is associated with thirty-five decision drivers out of the forty-four we discovered and thus represents a significant design decision. Since MLOps is a complex domain in its own right, it lies outwith the scope of this study to describe it in any more detail, but it shall be mentioned where appropriate for completeness and to emphasise its significance, such as when evaluating related decisions.

C. Automatic Integration and Delivery in an ML Context Decision

When deciding how to automate integration and delivery in a machine learning context, the simplest solution is to opt for no integration or delivery automation. Several practitioners strongly advise against this approach due to its detrimental impact on many forces, including process and work automa-
tion, iterative development, deployment velocity, dependable releases, reliability, independent development, modifiability, reproducibility, testability, auditability and continuous improvement.

An alternative option for continuous automation of the deployment processing is again to use a CI/CD pipeline, which was previously an option for the initial deployment decision. Our sources recommend this, particularly regarding the above forces. CI/CD pipelines are related to several other practices. For instance, they can be combined with a machine learning orchestrator to coordinate pipelines and experiments. An ML orchestrator is a component that can take responsibility for the automation of data set preparation and cleansing or for coordination of model development with respect to specific data sources and features.

Applying a CI/CD pipeline enables automated provisioning, that is, automation of the actual deployment step. The CI/CD pipeline can use a model registry to access models and model candidates. Pipeline parameters and versions can be stored in a machine learning metadata store where e.g. data and parameters can be tracked so that the practitioner can understand how specific models were built. It is worth noting that the two aforementioned practices entail a related design decision – How to store data used in machine learning applications – but this decision shall not be explored further since it is concerned with operational aspects of ML applications and no longer falls within the scope of this study.

An alternative solution is to manually perform automated deployments using build and deployment scripts, which is beneficial for process and work automation, iterative development, deployment velocity, dependable releases and reproducibility. The machine learning orchestrator (previously described in relation to CI/CD pipelines and for which two forces are described in the literature) is yet another potential decision option. Using an orchestrator allows monitoring data changes, which is beneficial for identifying whether data has changed over time. This can prove advantageous since it enables the data and model building pipelines to be triggered as early as possible based on changes to the data distribution (see Section IV-E), reducing the chance that sub-optimal models remain in production for longer than necessary. Process and work automation is also improved by using an orchestrator: by not running everything manually, we reduce the scope for human error and free up engineers for other tasks.

Two related practices (build and deployment scripts and CI/CD Pipeline) invite the practitioner to consider which tasks can be performed by a delivery pipeline or component. This decision shall be described in Section IV-D. Various other top-level decisions present themselves for consideration, e.g. how to trigger a machine learning pipeline or orchestrator, how to version data and which model versions should be deployed and how. These decisions shall likewise be described below. Finally, as shown in Figure 3, every decision involving automation also suggests considering MLOps as a design decision.

D. Delivery Tasks Decision

When planning how to deploy ML models, the practitioner needs to consider which tasks can be performed by a delivery pipeline or component, as presented in Figure 4. For this ADD, the list of forces in Table 3 for the corresponding decision options described below is empty (∅) because they are inherited from two parent decision options: CI/CD pipeline and build and deployment scripts.

One essential task that can be performed by a delivery pipeline or component is building, which, in the context of delivery tasks, may involve creating models, container images, executables, services or ML pipelines.

Another vital task is packaging, which involves taking build artefacts and suitably assembling them for further use. Packaging goes hand in hand with containerisation, which uses, e.g. Docker [21] to prepare application images and execute them in a containerised environment.

Testing is yet another delivery task and may include various techniques, such as canary testing (initially releasing new versions of an application to a subset of traffic/users), A/B testing (randomised, statistical comparison of versions) and hypothesis testing (a comparison of ML algorithms before deployment). The related decision of what to test in a machine learning pipeline also uses these techniques. This decision shall not be explored further since it is not concerned with deployment specifically. Deciding whether MLOps should be applied and, if so, when can be considered from the outset or when deciding what to test in a machine learning pipeline.

E. Trigger Pipeline or Orchestrator Decision

ML pipelines and orchestrators (“components”) can be initialised through triggers. Triggers vary in nature and may, for example, be based on events during the development process, based on feedback from a live system, be scheduled or manual. The corresponding ADD is how to trigger a machine learning pipeline or orchestrator, presented in Figure 5.

As in Section IV-D, the decision options’ forces for this ADD are inherited from a parent; in this case, the ADD how to automate integration and delivery in a machine learning context. Thus, the list of forces for the following trigger-specific decision options is empty (∅) in Table 3:

- **An on-demand trigger** initiates a pipeline or orchestrator via manual execution, such as by a human operator.
- **An on commit trigger** activates components following a commit event in a source control management or data version repository system.
- **An on schedule trigger** starts a component at a given time or with specified regularity.
- **If new training data** becomes available, this may also prompt a component to start.
- **If model performance degradation** is detected, this event may also serve as a component trigger. Note that model performance monitoring is a prerequisite for this option.
- **Changes to the data distribution** may trigger a component when statistical changes to any data set used to train
the model cross a predefined threshold.

When evaluating this decision and the model performance monitoring practice required for detecting model performance degradation, deciding whether MLOps should be applied and, if so, when can be considered at this stage, if the practitioner has not already done so.

F. Data Versioning Decision

Managing data is central to all phases of ML model development. Furthermore, deciding how to version data (presented in Figure 6) is directly related to model deployment since a significant design decision – discussed in Section IV-C – surrounds which model versions should be deployed and how.

Three decision options were identified in the literature, and it is crucial to consider these practices carefully since not all forces were described for each practice. For instance, handling large data files and reproducibility are defined for all three practices, whereas development velocity was defined for two practices and flexibility was only defined for one.

The simplest option is no data versioning, but practitioners consider this undesirable for handling large data files, which is not ideal since ML routinely uses large files. It is also detrimental with respect to reproducibility because it becomes highly challenging to keep track of changes made to data when practising either technique.

Data versioning in a code repository is a typical pattern that uses version control repositories such as Git9 to store ML data files. This may initially appear to be an appealing and intuitive approach since such repositories are excellent for versioning, e.g. source code. However, several drawbacks may be attributed to such systems when handling large files typically used in ML. Such code versioning systems are not suited to versioning large files and do not aid reproducibility since data file versions are hard to keep track of when code is updated simultaneously, and the development velocity is hindered because code versioning systems are slow when

9http://git-scm.com
**Packaging: Practice**

**Testing: Practice**

**Building: Practice**

**Deployment: Practice**

**Containerization: Practice**

**Canary testing: Practice**

**Hypothesis testing: Practice**

**A/B testing: Practice**

**Which tasks can be performed by a delivery pipeline or component? : Decision**

**What to test in a machine learning pipeline? : Decision**

**Should MLOps be applied and, if so, when? : Decision**

**Containerization: Practice**

**G. Deploying Model Versions Decision**

As mentioned previously, another significant ADD to consider is which model versions should be deployed and how. This ADD is depicted in Figure 7. The sources used in this study identified three main decision options, each related to two possible forces.

The decision to deploy a single model in production is the most straightforward option but is unsuitable if the practitioner is aiming for safe model transitions or rolling upgrades, primarily due to the necessary downtime involved in swapping out a single model version and the lack of availability of multiple model versions. The practice of rolling back to a previous model version, on the other hand, is more suited to both safe model transitions and rolling upgrades since the practitioner is easily able to revert to a previously-deployed model version. Finally, N versions in production is a pattern handling large files. Finally, the tight coupling with the same system used for source control reduces flexibility. This practice is considered unsuitable in the literature with respect to all four practices.

Our sources indicate that if the practitioner wishes to increase support for handling large data files, reproducibility and development velocity, using a data version repository designed to support large files in an ML context, such as DVC[^10], appears to be an ideal approach. Note that no impact on flexibility was documented in our sources when using a data version repository. Finally, the only related decision to consider, if not already decided, is whether MLOps should be applied and, if so, when.

[^10]: [http://dvc.org](http://dvc.org)
that is highly recommended in the literature for both safe model transitions and rolling upgrades because it enables smooth swapping out of model versions without downtime. This pattern is also suited to shadow mode testing, thus enabling A/B testing and canary testing, increasing confidence in new model versions being production-ready. Notably, N versions in production does not appear to impact reliability.

All three practices use a machine learning metadata store, which was previously encountered when considering how to automate integration and delivery in a machine learning context. The practitioner may also wish to consider MLOps if this has not been decided.

V. DISCUSSION

A. Discussion of the Research Questions

RQ1: After analysing thirty-five sources, we discovered evidence for seven ADDs, various relations between them, twenty-six decision options and forty-four decision drivers in the context of deployment for ML. We note that the decision options can be grouped into two main categories: automation-specific decisions and decisions that can be taken to enhance the benefits of automation.

Examples of automation-specific decisions are how to approach deployment of machine learning models, how to automate integration and delivery in a machine learning context and should MLOps be applied and, if so, when. Once a level of automation has been selected, the practitioner may consider other decisions, e.g. what the automation process should support, based on associated force impacts and specific requirements. For instance, the practitioner may consider which tasks they would like their delivery pipeline or orchestrator to perform and how to trigger it. Furthermore, the practitioner may consider how to version data and which model versions should be deployed and how. Opting for automation opens up new possibilities – many of which might not have even been considered if automation had been disregarded – that can aid ML considerably.

RQ2: A curious phenomenon involving one decision in particular emerged. MLOps appears multiple times as a related decision for all six other decisions. MLOps is so prevalent throughout the model that it may be worth considering MLOps first when planning the deployment. If they are identified, giving precedence to such recurring decisions makes sense since they potentially lead to solutions in multiple contexts. This discovery also indicates that MLOps may be a significant area for further research.

Similarly, when considering decision options, CI/CD pipeline appeared more than once. Interestingly, when considering CI/CD pipeline in the context of how to approach deployment, the practitioner is invited to consider how to automate integration and delivery, which also includes the CI/CD pipeline option and is a more comprehensive approach to automation. This represents the natural thought process of a practitioner searching for and choosing from solutions: upon considering a potential solution, the practitioner is led towards an all-encompassing approach that opens up new possibilities.

RQ3: Our research has helped us discover which decision drivers (forces) practitioners are most concerned with. When evaluating a given decision option, these are relevant factors and fall broadly into two categories: qualitative (e.g. modifiability and functional (e.g. iterative development). Forces (codes of which can be cross-referenced in Table [I]) can also be categorised as follows:

- Automation: f1.
- Development: f2, f4, f8, f9, f13, f15, f18, f35, f44.
- Understanding: f3, f10, f12, f32.
- Deployment: f5, f6, f16, f17, f29.
- Performance: f7, f22.
- Quality: f11, f20, f28.
- Support for ML-specific needs: f14, f23, f24, f25, f26, f27, f33, f36, f41, f42, f43.
- Business needs: f19, f21, f30, f31, f34, f37, f38, f39, f40.

The top three categories based on the number of forces were support for ML-specific needs, business needs and development. Out of all the decision options, using a CI/CD pipeline for the automated integration and delivery decision was mentioned most (in thirteen sources), whereas rolling back to a previous model when deciding which model versions should be deployed and how was mentioned least (in just two sources). For those decision options in Table [II] without force relations, the reader is reminded that any out-of-scope forces were excluded from the study and that these options inherit forces from their parent decisions and practices.
B. Threats to Validity

Stol et al. [22] describe method slurring, which encompasses common methodological errors that undermine GT and may generally occur when one claims “to use a research method without actually following its guidelines” – precisely (in the case of GT) when researchers “adopt an arbitrary subset of GT practices that are not recognisable as GT”. The likelihood of this study exhibiting method slurring is low since we include all the steps necessary for a GT study according to Stol et al. (simultaneous data collection and analysis, constant comparison, coding, memoing and theory development). We did not collect our data before beginning analysis; neither did we collect or categorise data according to existing theory, nor did we base our analysis on seed categories or preconceived analytical frameworks. We also specify the exact version of GT we applied (Straussian GT).

Our study is based on a subjective methodology, so it cannot be ruled out that our interpretation of the sources involved a certain degree of bias. This threat cannot be eliminated, but since both authors iteratively cross-checked each other’s activities, we believe any bias or inaccuracies were reduced.

A methodological threat to validity may arise when using unreliable or irrelevant sources or, conversely, excluding reliable or relevant sources. To mitigate this threat, we continued to code sources until we reached theoretical saturation to reduce the impact of outliers.

Hagstrom et al. [23] and Piasecki et al. [24] describe using Web search engines for finding sources when conducting literature reviews. Any search-based bias was mitigated by consistently applying such search methods and inclusion-exclusion criteria and allowing our findings to guide us.

GT aims to discover and explain phenomena that exist – it neither claims to exhaustively capture all such phenomena, nor does it claim to quantify their frequency or guarantee a specific distribution or coverage [25]. We similarly make no such claims, nor do we claim to document best practices or universally applicable recommendations, since the modelled practices and decisions are merely observed phenomena.

Considering every possible source would have been infeasible and would have likely brought little additional benefit because we coded to theoretical saturation. For the same reason, it is unlikely that a multivocal literature review would have led to dramatically different results. Thus, we are confident that our results accurately model practitioner understanding of the domain.

VI. CONCLUSIONS

This GT-based grey literature study modelled ADDs, decision options, relations and decision drivers in SE/SA for ML within the context deployment. Our findings uncovered various significant interrelated design decisions when deploying for ML. We identified decisions involving automated integration and delivery, pipeline triggering, model version deployment, delivery pipeline component tasks, testing within a pipeline, data versioning and MLOps, with each decision being associated with various options that may be selected. The resulting UML-based model can be used by evaluating the specific decision drivers associated with each option for a given decision to guide practitioners and advances the scientific understanding of typical practices. Potential applications of our ADDs could also be to reduce uncertainty or complexity in the ML decision space, assess conformance to established architectural practices, or identify anti-patterns in existing systems. Our work forms the basis for continued research in this domain.

ACKNOWLEDGEMENT

This work was supported by the FFG (Austrian Research Promotion Agency) project AMMONIS (no. 879705).
REFERENCES


