Machine Learning Systems in Industry: Challenges and Practices

PEGAH SALEHI

This report explores the various challenges and practices involved in deploying machine learning (ML) systems within the industry. It systematically covers various stages of the ML deployment workflow, including data management, model learning, testing, deployment, and cross-cutting aspects such as optimization and end-user trust. It emphasizes the importance of a holistic approach that balances technical excellence with ethical considerations. Moreover, the report also provides a deeper understanding of the complexities involved in developing robust, scalable, and secure ML systems, providing a foundation for future research and innovation in ML deployment practices.

1 INTRODUCTION

In today's digital age, machine learning (ML) is transforming how we solve problems and make decisions across various industries. From healthcare diagnostics to social media algorithms and self-driving cars, ML systems are becoming an integral part of technological advancements. However, the integration of ML into commercial software introduces a unique set of challenges. Unlike traditional software, ML systems learn from data, requiring a change in how we develop, deploy, and maintain these systems. This change demands collaboration between experts from different fields, including data science, software engineering (SE), and more, to ensure the systems are effective, reliable, and ethical.

The complexity of developing and deploying large-scale ML systems in industrial settings arises from the dynamic environments in which these systems operate, interacting with various components and adapting to new data. Ensuring these systems meet high standards of adaptability, scalability, privacy, and safety is paramount, especially in critical applications affecting people's lives. Addressing these challenges requires not only rethinking established SE practices but also adopting new approaches tailored to the ML lifecycle.

To navigate this complexity, the Machine Learning Technology Readiness Levels (MTRL) framework [8] offers a systematic approach for developing, deploying, and ensuring the robustness of ML and artificial intelligence (AI) systems. Drawing from disciplined processes used in engineering systems, such as spacecraft systems engineering, MTRL creates a structured development pathway for ML technologies. It defines a series of stages—each with specific goals, documentation requirements, and review processes—to guide and measure the maturity of ML systems from initial research to deployment. By incorporating key distinctions from traditional SE and emphasizing data considerations at each stage, MTRL aims to foster reliable, responsible, and efficient development of ML technologies.

This report seeks to shed light on the challenges of ML system deployment in commercial and industrial contexts, synthesizing SE hurdles and solutions. Through a systematic review of existing literature and incorporating insights from the MTRL framework, we identify key areas needing attention and propose solutions to address them. Our work aims to bridge the gap between traditional SE and ML development practices, offering a guide to navigating the complexities of ML system deployment. This report is not only a compilation of challenges and solutions but also a call to action for industry professionals and researchers. It emphasizes the need for continuous innovation, ethical consideration, and interdisciplinary collaboration to develop ML systems that are not only technologically advanced but also socially responsible. It also highlight the importance of

Author's address: Pegah Salehi, pegah@salehi.no.

adaptability, scalability, privacy, and safety as critical quality attributes in ML system development, providing a roadmap for future research and practice in this rapidly evolving field.

2 DATA MANAGEMENT

Data is crucial to ML success, with its quality affecting the solution as much as the algorithm used. Creating quality datasets is typically the first step in ML production pipelines, often consuming more time and energy than expected due to various challenges. This section describes challenges that the machine learning community encounters during data management, as illustrated in Figure 1.

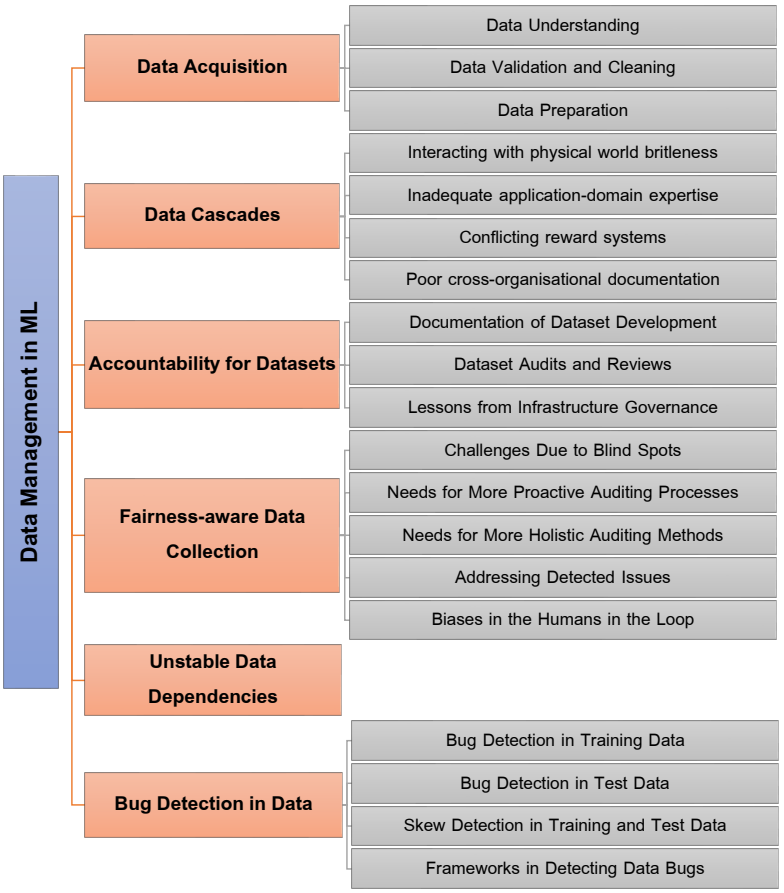

Fig. 1. Challenges in Data Management

2.1 Data Acquisition

There are various data management challenges that arise at different stages of the data lifecycle, as described below [13]:

Data Understanding: Understanding ML data is foundational for developing accurate and fair models that involve two key stages: initial sanity checks and ongoing analyses during model iteration. Initially, it's essential to verify that the data has the correct "shape" and characteristics, such

as appropriate ranges for numerical features and expected distributions for categorical variables. This phase ensures the data is suitable for training a model without obvious flaws. Through iterations, the model is refined to examine feature-based impacts on accuracy, detect disparities between training and serving data (known as skew), and explore the data's lifecycle, including dependences and errors.

For this purpose, it is essential to use advanced tools and methodologies; for example, SQL allows for straightforward data integrity validations. ProvDB offers detailed insights into the data's origins and transformations, highlighting dependencies and potential biases. MLCube facilitates the examination of data slices to pinpoint where models excel or falter, enabling targeted improvements. Together, these methodologies form a comprehensive approach to understanding and refining ML data, ensuring models are built on a solid foundation of accurate and representative information.

Data Validation and Cleaning: This phase relies heavily on the quality of their input data. If data is inconsistent, such as variations in the spelling of country names, or if a feature's measurement unit changes unexpectedly, it can lead to incorrect model predictions. To prevent these issues from affecting model training, data must be checked and corrected early on. This can include standardizing text formats, creating new fields for unforeseen features, or diagnosing missing features through root cause analysis. However, not all issues can be fixed automatically, so it's crucial to have clear guidelines, or "playbooks," for manually addressing these problems. Moreover, managing alerts for data issues involves balancing the sensitivity of alerts to avoid overwhelming users with false alarms while still capturing significant errors.

Data Preparation: Data preparation is a critical yet complex step in ML that involves creating and selecting the right features to train models effectively. Initially, it focuses on identifying features that strongly predict the desired outcome, and as models evolve, it shifts towards optimizing resources and reducing latency by choosing a subset of features without compromising accuracy. This process also includes enhancing training data with additional information from other sources when the original data is incomplete. Techniques like normalization, bucketization, and one-hot encoding are commonly used to transform raw data for better ML model performance. Furthermore, the choice of features and their transformations can significantly impact model accuracy, leading to continuous experimentation with adding or removing features to improve outcomes. Data enrichment, involving augmenting data with external information, plays a crucial role in improving model accuracy. However, the process is challenging, requiring careful consideration of data quality, the cost of obtaining data, and the potential trade-offs between more data and improved data quality.

2.2 Data Cascades

Data Cascades are defined as negative, compounding events stemming from data-related issues within AI systems, leading to technical debt over time. These cascades are influenced by the actions of those involved in AI development, the environment where the AI operates, and often result from applying conventional AI practices in complex, high-stakes domains. The cascades are hard to detect, can be triggered by various factors including poor data practices, and have significant negative impacts on both AI development and deployment, sometimes causing further cascades and always generating technical debt. They are typically avoidable with early and systematic interventions in the development process, despite challenges such as undervaluing data, data scarcity, and dependency on partners. The following is a list of data cascades and associated behaviors, sorted by frequency [14].

Interacting with physical world britleness: AI systems face challenges when transitioning from digital to physical environments, particularly in high-stakes areas like health care, road safety, and environmental sensing. These challenges stem from the "brittleness" of AI systems, meaning they struggle to adapt to the unpredictability and complexity of the real world. This difficulty arises due to several factors, including limited training data, rapidly changing environments, and complex phenomena that the systems were not initially designed to handle. As a result, AI models can fail when encountering small changes in their operational context, such as shifts in camera position due to weather, leading to significant errors like misidentifying traffic violations or health issues. These failures, known as "data cascades," often emerge from hardware malfunctions, environmental changes, and shifts in human behavior, taking years to become apparent and frequently resulting in the abandonment of projects. Efforts to improve model robustness include monitoring data sources for unexpected changes, introducing noise to training data, and enhancing data literacy among system operators.

Inadequate application-domain expertise: AI practitioners lack expertise in the specific areas they're applying AI to, like healthcare or wildlife conservation. This lack of domain knowledge leads to "data cascades," where errors compound over time due to poor data handling and decision-making. For example, incorrect predictions about poaching locations were made because the AI didn't include all relevant data, leading to inefficient use of limited resources. The problem is exacerbated when practitioners have to make complex decisions without the input of domain experts, resulting in errors and inefficiencies that are costly to correct. This underscores the importance of involving experts throughout the AI development process to ensure data is accurately collected, interpreted, and applied.

Conflicting reward systems: Misaligned incentives and priorities among AI practitioners, domain experts, and field partners can also lead to data cascades. For example, incorrect GPS settings disrupted a wildlife conservation project's ML model. Challenges include practitioners viewing data collection as a non-technical task, budget constraints leading to inadequate compensation for data collectors, and a general lack of data literacy among field partners. This misalignment often results in costly iterations, project abandonment, or the need to find alternative data sources.

Poor cross-organisational documentation: The problem of missing documentation and metadata in cross-organizational data sharing, leading to "data cascades" where incorrect assumptions about data lead to significant losses, such as months of medical robotics data. This issue arises from the undervaluation of data documentation and a lack of awareness about the need for high-quality data. The consequences include wasted effort, the inability to build accurate models, and the need to discard or redo data collection. Successful cases involved detailed data management plans and the recording of comprehensive field notes to ensure data usability and reliability.

2.3 Accountability for Datasets

Accountability is characterized by the answerability of actors for outcomes, which necessitates a thorough examination of who knows what information and how this information is utilized. The framework for accountability involves three phases: the sharing of information, deliberation and discussion, and the imposition of consequences by a forum. This discussion is particularly focused on the first phase—information sharing—which is critical for enabling meaningful dialogue and the potential for consequences. Although sharing information is necessary for accountability, it is not sufficient on its own. It calls for a detailed understanding of the roles and responsibilities of

Table 1. Stages of the Data Development Lifecycle [6].

Requirements analysis	Deliberations about intentions, consultations with stakeholders, and analysis of use cases determine what data is required.
Design	Research is performed and subject matter experts are consulted in order to determine whether the data requirements can be met, and if so how best to do so.
Implementation	Design decisions are transformed into technologies such as software systems, annotator guidelines, and labeling platforms. Actions may employing and managing teams of human expert raters.
Testing	Data is evaluated and decisions about whether or not to use it are made.
Maintenance	Once collected, a dataset requires a large set of affordances, including tools, policies and designated owners.

various actors in dataset development, including the need for transparency about which actors bear responsibility for reliability and the integrity of datasets. This approach suggests a shift towards treating dataset development as an engineering discipline, where documentation, oversight, and maintenance are fundamental components. By adopting practices from SE and infrastructure management, Table 1 proposes a structured lifecycle for dataset development that includes stages such as requirements analysis, design, implementation, testing, and maintenance [6]. This structured approach aims to improve the visibility, traceability, and quality of datasets, thereby enhancing their accountability and reliability in ML applications.

Documentation of Dataset Development: The documentation of the dataset development process is crucial for ensuring transparency, accountability, and quality in ML systems. [6] presents a comprehensive framework for systematically documenting every stage of the dataset development lifecycle. This framework divides the dataset development process into five key stages: Data Requirements Accounts, Dataset Design Accounts, Dataset Implementation Accounts, Dataset Testing Accounts, and Dataset Maintenance Accounts. Each stage requires specific documentation artifacts, with designated owners responsible for various roles and duties. For instance, the Data Requirements Accounts focus on justifying the use of data as a solution, considering both intended and unintended consequences, and employing the 6 W's (who, whom, what, where, when, and why) to detail scenarios. This stage aligns with requirements analysis in engineering, advocating for Data Requirements Specifications that encompass both quantitative and qualitative factors. Similarly, the Dataset Design Accounts stage delineates the means of achieving the documented requirements, necessitating research of existing datasets and thorough justification of design decisions. This structured approach to documentation not only facilitates the clarity and traceability of dataset development processes but also encourages critical reflection and engagement with stakeholders to resolve conflicts and make transparent the values embedded in the data.

Moreover, the framework highlights the importance of documenting the implementation, testing, and maintenance phases, with each phase requiring detailed accounts that explain the decisions made and their justifications. For example, Dataset Implementation Accounts are likened to code comments, focusing on the rationale behind specific implementation choices. Dataset Testing Accounts address the necessity of rigorous testing to evaluate the dataset's fitness for use, including requirements testing and adversarial testing to uncover potential harms. Lastly, the Dataset Maintenance Accounts emphasize the ongoing need for corrective, adaptive, and preventive maintenance to ensure the dataset remains relevant and reliable over time. This comprehensive documentation strategy underscores the multifaceted nature of dataset development as an engineering discipline, requiring meticulous planning, stakeholder engagement, and iterative refinement to address the

dynamic challenges and ethical considerations inherent in ML research.

Dataset Audits and Reviews: It outlines a structured approach to auditing, which includes phases of scoping, mapping, artefact collection, evaluation, and reflection, with a focus on analyzing datasets and models in accordance with organizational policies and commitments. This process leverages the comprehensive documentation created during dataset development as an audit trail, thus facilitating auditability by design. Additionally, it highlights the necessity of ongoing reviews throughout the dataset lifecycle—including requirements and design reviews—to ensure datasets remain relevant and trustworthy as contextual circumstances evolve, reinforcing the commitment to transparency and accountability in dataset utilization.

Lessons from Infrastructure Governance: This strategy discusses how managing datasets is similar to managing public infrastructure projects, where both often start with a lack of understanding of the potential risks and the amount of work needed. It suggests that, just as with public projects, dataset projects should prioritize anticipating and managing risks. In [6] is recommended to involve stakeholders early in the dataset creation process to help identify and address potential issues, highlighting the importance of making decisions in a way that considers the welfare of all involved. This could involve either a top-down approach, where decisions are made based on maximizing benefits for the most people, or a more consensus-driven approach, ensuring solutions are acceptable to all stakeholders. It also mentions the importance of long-term accountability, noting that problems in datasets may not become apparent for years, and stresses the importance of including diverse voices and perspectives in decision-making processes to ensure fairness and legitimacy.

2.4 Fairness-aware Data Collection

While the traditional focus is on developing algorithms to mitigate biases, industry professionals increasingly recognize the need to address fairness at the data collection and curation stage, with many attempting to collect more diverse data as a solution. However, challenges remain, such as a lack of structured processes for ensuring balanced data collection, and a need for better communication between those collecting data and those developing ML models. Tools and processes that can guide fair data collection are seen as valuable, with suggestions for targeted strategies to engage underrepresented user groups and the design of test sets that can detect potential biases. This reflects a shift towards prioritizing high-quality, representative data collection to support fairness in ML applications, suggesting future research should focus on developing tools and methods that facilitate these objectives [5].

Challenges Due to Blind Spots: There are several difficulties teams face when it comes to creating fair ML systems due to their blind spots in understanding and recognizing all forms of unfairness. It highlights the challenges in collecting and curating data to represent diverse subpopulations adequately, which is crucial for preventing bias in ML applications. Teams often don't realize the full extent of fairness issues until they receive negative feedback from users or media. The practice of brainstorming potential problems and including fairness-focused evaluations during hiring processes is mentioned as a way to combat biases. However, no single team possesses all the necessary cultural or domain-specific knowledge to identify every potential bias, underscoring the need for diverse teams and the sharing of knowledge across organizational boundaries. It also touches on the difficulty of obtaining additional data to correct biases, using the example of a team struggling to recognize celebrities from different countries to illustrate the challenge. It suggests

that future research could explore methods for facilitating the exchange of knowledge and resources across teams to better address fairness issues in ML systems.

Needs for More Proactive Auditing Processes: It outlines the need for more proactive and systematic approaches to auditing fairness in ML systems. Currently, many teams rely on reactive methods, addressing fairness issues only after receiving customer complaints, which contrasts sharply with the proactive measures used for detecting potential security risks. This approach often leads to the late discovery of biases, and the majority suspect there are still undetected biases. Moreover, efforts to address fairness are not widely recognized or rewarded within organizations, with a small percentage of teams prioritizing fairness significantly. This has led to calls for more efficient monitoring tools and strategies to convince team members of the importance of addressing fairness issues, highlighting the need for support in making fairness a fundamental part of the ML workflow and reducing manual labor in fairness monitoring.

It is necessary to develop domain-specific auditing processes, implement scalable and comprehensive auditing methods, and develop mechanisms to enable auditing without access to individual demographic information as a result of legal and ethical restrictions. There's an urgent call for resources that include metrics, tools, and educational materials tailored to specific application domains to help practitioners navigate the complexities of fairness in ML. This includes developing processes and tools for fairness auditing with only coarse-grained demographic information and exploring human-in-the-loop approaches that combine automated methods with human judgment. Overall, there is a significant gap in current fairness auditing practices in ML, with a strong demand for more proactive, systematic, and domain-specific approaches to ensure fairness across diverse application areas.

Needs for More Holistic Auditing Methods: It emphasizes the necessity for broader and more inclusive auditing methods in fairness for ML systems, especially those that engage in complex interactions with users, such as chatbots, web search, and adaptive learning technologies. Unlike areas like recidivism prediction or face recognition, where fairness can be gauged through straightforward metrics like error rate parity across groups, applications with richer user-system interactions require a more nuanced understanding of fairness. It also points out the importance of considering the overall impact of ML systems on various user groups rather than just focusing on individual model metrics, the challenge of aligning system behavior with diverse user expectations, and the ethical dilemmas of modifying system outputs to address fairness concerns. There's a call for simulation-based approaches to better understand and mitigate potential fairness issues in systems involving complex user interactions, suggesting that future fairness auditing should encompass system-level evaluations and user impact studies to truly address the multifaceted nature of fairness in real-world applications.

Addressing Detected Issues: Teams face significant challenges when trying to address fairness issues in ML systems. They struggle to find cost-effective solutions and often can't pinpoint the exact causes of fairness problems, especially with "black box" ML models. There's a need for guidance on choosing the best strategies, whether it's changing the model, adjusting data, or other interventions, with a majority expressing a desire for better support in strategy comparison. Additionally, there's concern over the unintended consequences of fairness interventions, with some fixes inadvertently introducing new biases. Estimating the amount of additional data needed to address fairness issues is largely guesswork, leading to costly trial and error. There are also ethical dilemmas about the influence of technical choices on society and the fairness of targeting specific subpopulations for data collection. Ultimately, many fairness issues may be best addressed

through broader system design changes rather than focusing solely on algorithmic adjustments, emphasizing the importance of minimizing harm and considering the real-world impacts of fairness interventions.

Biases in the Humans in the Loop: This part highlights concerns about biases present in humans involved in ML development processes, such as those annotating training data or participating in user studies. It mentions the need for strategies to identify and mitigate these biases. For instance, an automated essay scoring team suspects that scorers' judgments may be influenced by irrelevant factors and suggests auditing this process by testing whether changing the style of an essay without altering its content affects its score.

2.5 Unstable Data Dependencies

Unlike code, where dependencies are easily tracked, data dependencies are harder to spot and can lead to complex issues that are tough to resolve. When systems use data from other sources that change over time, it can cause problems that are expensive to fix. For instance, if an external data source updates without the ML system knowing, it might harm the system's performance. To prevent issues with changing data, one strategy is to freeze data at a certain version, but this can lead to its own problems, like outdated information and maintenance costs. Additionally, ML systems often use data that doesn't add much value, making them unnecessarily complex and vulnerable to issues. Identifying and removing these unhelpful data dependencies is crucial but challenging, and there's a lack of tools to help manage these data relationships effectively, unlike in traditional SE where such tools are common [15].

2.6 Bug Detection in Data

Data quality significantly influences the system's performance, making bug detection in data a crucial aspect of ML testing. Early identification of bugs is essential because inaccuracies in the data can propagate through the system, creating a feedback loop that exacerbates even minor errors over time. The visibility of data generation logic is often limited within the ML pipeline, and storing data in formats like CSV may remove important semantic information, making bug detection more difficult [19].

Bug Detection in Training Data: To ensure a ML system performs well, it's crucial to examine the training data for errors. Tools like data linters help spot issues such as miscoding (e.g., mistyping a numeric value as a string), outliers, and data organization problems. Metrics can also gauge whether training data adequately cover all necessary scenarios. Furthermore, a technique called MODE tests training data by identifying "faulty neurons" in neural networks that cause errors, significantly improving test effectiveness.

Bug Detection in Test Data: Test data can contain adversarial examples that pose security risks by misleading the ML model into making incorrect predictions. Techniques for detecting such adversarial examples include augmenting deep neural networks (DNNs) with sub-networks that identify genuine data versus data with adversarial perturbations and using model mutation to expose sensitive adversarial samples. Despite these efforts, detecting adversarial examples remains a challenging task, as existing detection methods can often be bypassed.

Skew Detection in Training and Test Data: It's crucial for the consistency and performance of ML models that the training and test data exhibit similar characteristics and distributions. Skew

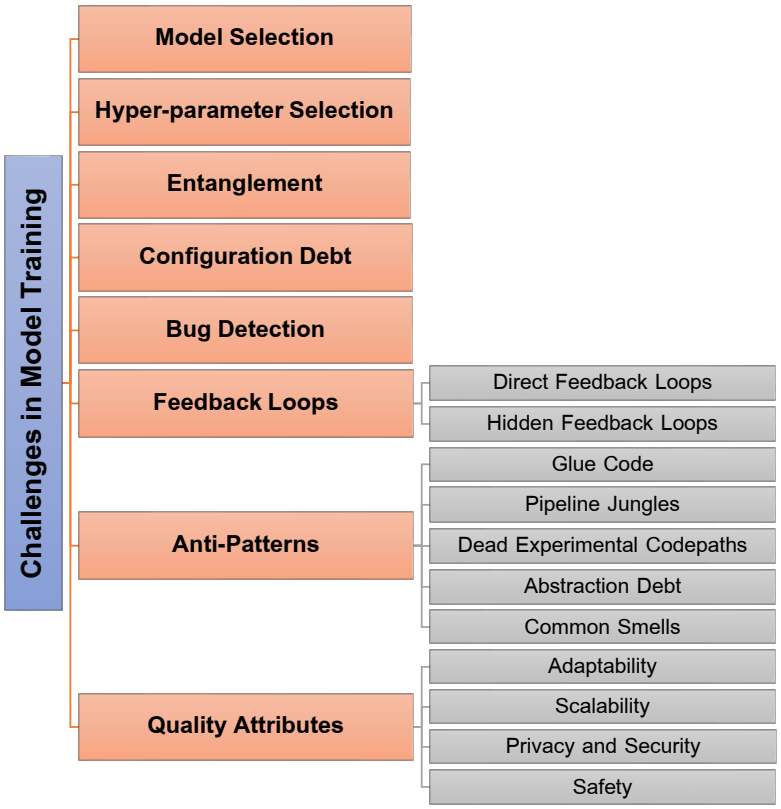

Fig. 2. Challenges in Model Learning

detection focuses on identifying discrepancies between these datasets, using methods like Kernel Density Estimation and vector distance measurements to approximate the likelihood of the model encountering similar inputs during training. Effective skew detection helps in identifying and quantifying differences in features and distributions between training and serving data.

Frameworks in Detecting Data Bugs: Frameworks for detecting data bugs apply various constraints and measures to identify issues within datasets, such as unexpected string values, missing columns, and new feature columns. Systems like TFX incorporate these frameworks to facilitate early bug detection and debugging. Moreover, frameworks like ActiveClean and BoostClean involve iterative data cleaning and domain value violation checks to enhance model performance, indicating the importance and efficacy of structured approaches in maintaining data quality and model reliability.

3 MODEL LEARNING

Model learning is a crucial phase in the ML deployment process, receiving significant focus from the academic community. Despite this growing attention, there are numerous practical challenges that need to be addressed during this stage, as discussed below and shown in Figure 1

3.1 Model Selection

In practical situations, the choice of a ML model is often influenced by its complexity. While complex models like those used in deep learning (DL) are popular in research, simpler models are usually preferred in real-world applications. These simpler models, such as shallow neural networks, Principal Component Analysis (PCA), decision trees, and random forests, are favored for several reasons. They can quickly demonstrate a concept, facilitate the setup of an end-to-end system, and provide valuable feedback without the complexity that can hinder development and deployment. This approach not only speeds up the deployment process but also ensures that the system remains manageable and efficient, particularly in environments with limited computational resources. For example, simpler models were used in the AirBnB search system and onboard scientific instruments for the Europa Clipper spacecraft due to their efficiency and lower demands on computational power. Additionally, the interpretability of models like decision trees makes them attractive for applications where understanding the model's decisions is crucial, such as in banking for churn prediction. Despite the advantages of simpler models, DL continues to be applied in areas like unmanned aerial vehicles (UAVs) due to its superior data processing capabilities, although its computational demands limit its deployment in resource-constrained environments [12].

3.2 Hyper-parameter Selection

Hyper-parameter selection involves choosing optimal settings for a model's hyper-parameters, which are parameters not learned from data but set before the training process. This includes things like the depth of a decision tree or the number of hidden layers in a neural network. Hyper-parameter optimization (HPO) is crucial because it involves multiple training cycles to find the best hyper-parameter settings, making it a resource-intensive and expensive process, especially for DL applications. The complexity of HPO tasks can grow exponentially with each additional hyper-parameter, complicating the optimization process. Practical challenges also include defining a complete search space due to limited problem knowledge and adapting HPO to specific hardware constraints, such as those in mobile devices, to balance model accuracy with resource limitations [12].

3.3 Entanglement

Complex ML models challenge traditional software practices that emphasize clear boundaries and isolated changes, as these models inherently blend data and behavior in ways that are difficult to separate. This blending, known as entanglement, means that changes in one part of the system (like input features, hyper-parameters, or learning settings) can unpredictably affect others, following the principle that changing anything changes everything (CACE). Strategies to mitigate these issues include using model ensembles and detecting changes in prediction behavior, but these too have their complications. Additionally, when models are built upon each other to correct minor discrepancies or adapt to slightly different problems, a dependency cascade can occur, locking the system into a rigid structure that's hard to improve. Moreover, when models' outputs become inputs for other systems without clear declarations, it creates hidden, tight couplings across the system, making any change potentially disruptive and costly. This complex interdependency and lack of clear boundaries significantly increase technical debt, making ML systems challenging to maintain and evolve [15].

3.4 Configuration Debt

Configuration in ML systems is a critical aspect that, if overlooked, can lead to significant technical debt, much like overlooked code can in traditional software development. As these systems grow and evolve, configurations—which define features, data selection, learning settings, and more—become complex and hard to manage, outnumbering lines of code and hiding potential errors. Proper management of configuration is essential to avoid costly mistakes that waste time and resources. This includes practices such as making configurations easily modifiable, reducing manual errors, visually comparing configurations, and ensuring configurations undergo thorough review processes [15].

3.5 Bug Detection

Bug detection in learning programs is about ensuring that ML algorithms are correctly implemented and configured. This includes verifying the proper design of the model architecture and identifying any coding errors. Various methods are used for this purpose, such as ML unit testing with TensorFlow to check if the code functions as expected, testing learning algorithms with unit tests for stochastic optimization, and examining algorithm configurations for compatibility issues across different operating systems, languages, and hardware. Moreover, the selection of algorithms is critical, with studies showing that traditional algorithms can sometimes perform as well as, or better than, DL at a lower cost. Mutant simulations and static analysis tools are also employed to identify faults in the learning program by simulating errors and analyzing tensor behavior in frameworks like TensorFlow [19].

3.6 Feedback Loops

Live ML systems have a unique characteristic where they might influence their own updates over time, creating a complex situation called "feedback loops." This makes it hard to predict how a model will act once it's out in the world because these loops can subtly change the model's behavior, especially if updates are not frequent. There are two main types of feedback loops: direct and hidden. Direct loops happen when a model influences the data it will learn from in the future, often leading to the use of less ideal algorithms due to scalability issues. To lessen this impact, strategies like adding randomness or segmenting data are used. Hidden loops are trickier; they occur when two systems affect each other indirectly, like two separate systems on a website influencing user interactions without direct connection, or even two stock-market prediction models from different companies affecting each other's outcomes. These loops, especially the hidden ones, present a challenging puzzle for ML researchers due to their indirect nature and the broad implications they can have [15].

3.7 Anti-Patterns

In ML systems, a minimal portion of the code is for learning or prediction, with the bulk being "plumbing." Such systems often develop high-debt design patterns, including system-design anti-patterns that ideally should be avoided or improved for better system efficiency and maintainability [15].

Glue Code: In ML systems, developers often use generic, open-source packages to build their applications. This approach, however, leads to a design pattern called "glue code," where a significant amount of custom code is written to integrate these generic packages into the system. While initially appearing efficient, this approach results in systems that are hard to adapt or improve due

to their heavy reliance on the peculiarities of specific packages. Over time, systems become so intertwined with these packages that changing them becomes nearly impossible without significant effort. An effective strategy to combat this issue is to encapsulate these packages within common APIs, making the system more flexible and reducing the reliance on any single package.

Pipeline Jungles: Pipeline jungles are a specific form of glue code that emerges during the data preparation phase of ML projects. They consist of complex, hard-to-manage networks of data processing steps, including data scraping, joining, and sampling, which often result in a chaotic and inefficient infrastructure. These jungles make it difficult to manage data pipelines, detect errors, and recover from failures, significantly increasing the technical debt of a system. Avoiding pipeline jungles requires a holistic approach to data collection and feature extraction, and sometimes a complete redesign of the data preparation process to streamline operations and foster innovation.

Dead Experimental Codepaths: Over time, ML projects might accumulate "dead" code that was only used for experiments but never removed. This leftover code can make the system more complex and harder to maintain, similar to how clutter in a home makes it harder to clean and organize. Occasionally, it's important to clean up these unused code paths to prevent bugs and reduce complexity, much like decluttering a house.

Abstraction Debt: Abstraction debt arises from a lack of clear, high-level structures or models in ML systems that define how components interact and operate. This is similar to having a toolbox without the right tools for specific jobs, making tasks more difficult and less efficient. The absence of widely accepted standards for these abstractions, especially for distributed learning, leads to confusion and inefficiencies in system design.

Common Smells: "Smells" in ML systems are subtle indicators of deeper issues, reflecting poor design choices or practices. For instance, relying on basic data types to represent complex information, using multiple programming languages within a single system, or frequently resorting to prototyping are all smells that suggest underlying problems. These practices can make systems harder to test, maintain, and scale, indicating a need for better abstractions, interfaces, and a more thoughtful approach to system design to ensure robustness and scalability.

3.8 Quality Attributes

Adaptability: Training a ML model starts with clearly defining the problem it's meant to solve, which can be tricky because problems are often presented in vague terms. It's essential to translate these problems into specific ML tasks, which isn't always straightforward. Early assumptions can lead to oversimplified approaches for complex issues, resulting in less effective solutions. For example, LinkedIn first analyzed the pros and cons of sending automated emails to users to better define their ML problem, aiming to optimize email notifications. Moreover, preparing the data for the ML model, known as feature engineering, is a detailed and time-consuming process because the raw data isn't immediately usable and must be processed first. This task is complicated by the diverse nature of the data and its relation to the predictions the model will make. Additionally, limitations in ML technology require the development of custom tools for successful application, and experimenting with ML models can lead to messy, hard-to-maintain code if not managed carefully. Overall, setting up an ML model involves careful problem definition, data preparation, and managing the technical challenges of applying ML algorithms [11].

Scalability: In the world of ML, when models get very big and complex, it's like that giant puzzle. So, instead of using just one computer to train the model (which would take a very long time), we use many computers to work on different parts of the model at the same time. This makes the whole process faster, but it also brings new challenges, like making sure all the pieces fit together perfectly in the end [11].

Privacy and Security: When companies train ML models, they often use sensitive data that shouldn't be shared openly. This makes it hard for engineers to look at the data closely to understand it better. Different countries have different rules about keeping data private, which makes building models that respect privacy even more complicated. Privacy and security issues manifest through four critical types of attacks: model extraction, model inversion, adversarial, and poisoning attacks. Model extraction attacks aim to replicate the ML model's parameters, compromising its confidentiality and intellectual property. Model inversion attacks, on the other hand, focus on deducing sensitive information from the model's outputs. Adversarial attacks involve inputting subtly modified data to deceive the model into making incorrect predictions, while poisoning attacks corrupt the training data, leading to a compromised model. These security challenges highlight the necessity for robust defense mechanisms to protect ML models and their data from malicious exploits. To address privacy concerns in ML, several defensive solutions have been proposed, focusing on differential privacy, homomorphic encryption, secure multi-party computation, trusted execution environments, and Federated learning [10].

- Differential privacy techniques aim to add noise to the data or model outputs, ensuring that the privacy of individual data points is maintained while allowing for useful aggregate data analysis.
- Homomorphic encryption allows computations to be performed on encrypted data, providing results without ever exposing the original data.
- Secure multi-party computation enables a group of parties to jointly compute a function over their inputs while keeping those inputs private.
- Trusted execution environments offer a secure area within a main processor to protect code and data from disclosure or modification.
- Federated learning is a method used to enhance privacy by performing calculations directly on the devices that generate the data, rather than moving the data to a central location for analysis. This approach helps solve a major privacy issue by ensuring that sensitive information does not have to leave the user's device. It includes a technique called secure aggregation, which allows updates from individual devices to be combined without exposing the data from any single device. This system has been successfully used in large-scale projects, such as improving the functionality of smartphone keyboards, showcasing its practical application in everyday technologies. This approach, however, faces challenges such as bias, as not all devices have equal opportunities to participate in the training process due to network connectivity and charging requirements. Additionally, deployment may be constrained to specific devices with certain capabilities, introducing potential biases.

These approaches, while increasing computational overhead and requiring customization, offer promising pathways to safeguard privacy in the crucial stages of data preparation and model training in ML.

Safety: For systems that need to be very safe, like those in cars that help drivers or avoid accidents, making sure the ML models they use are reliable is super important. However, because these models learn from data and can change over time, ensuring they are always safe is tricky.

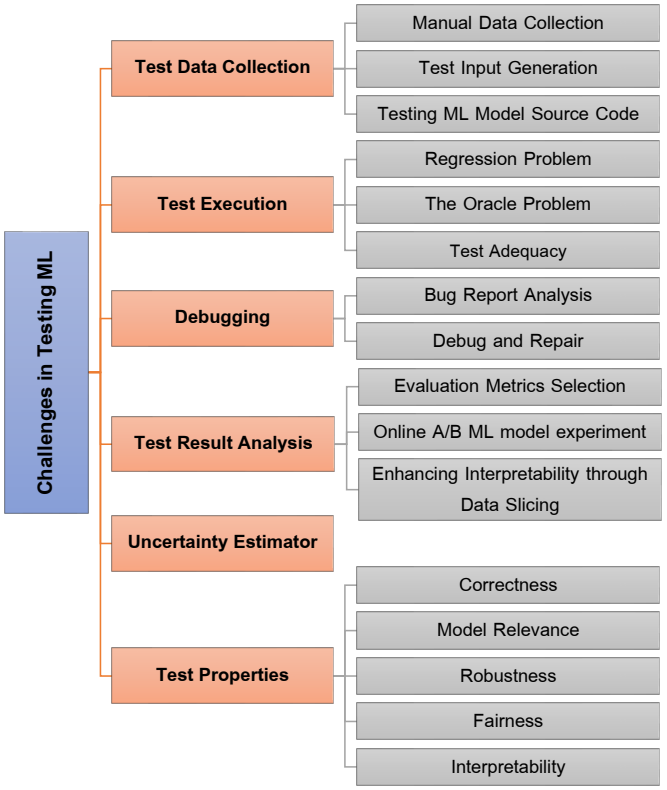

Fig. 3. Challenges in Testing ML

Current safety rules might not fully cover the new ways these models work. It's like creating a new game but still trying to follow the old rules that don't quite fit. So, there's a big effort to develop new ways to make sure these models can be trusted to make the right decisions, especially in situations where safety is crucial [11].

4 MODEL TESSTING

The concept of ML testing is introduced as an activity aimed at detecting discrepancies, referred to as "ML bugs," between the existing and required conditions of a ML system. These bugs could exist in any component of the ML system, including the data, learning program, or framework. The dynamic nature of ML systems may change over time with new data, which makes the testing process more complex compared to traditional software testing.

Figure 3 structures of challenges in ML testing. This includes both offline and online testing phases. Offline testing involves evaluating the ML model with historical data before deployment to ensure it meets the required conditions without the real application environment's unpredictability. In contrast, online testing occurs after deployment, focusing on the model's performance in real-time, capturing user interactions, and addressing the limitations of offline testing by adapting to new and unforeseen data and scenarios.

4.1 Test Data Collection

Given the data-driven approach of ML, it is crucial for professionals in the field to compile a comprehensive dataset for testing to assess the various attributes of a ML model effectively.

Manual Data Collection: Collecting test inputs and golden labels by hand is labor-intensive and costly but crucial for ML system development due to the need for high-quality test data. It typically involves domain experts or annotators, which consumes significant time and resources. To improve efficiency, four main methods are employed:

- Active Learning Approach involves selectively labeling the most informative data points instead of randomly labeling all available data. This method prioritizes data that the model is currently uncertain about, making the labeling process more efficient and cost-effective. This approach leverages the model's predictions to identify which data points, if labeled, would most improve the model's performance.
- Scenario-Based Collection involves defining typical usage scenarios based on customer requirements and collecting test data specifically for each scenario. This targeted approach ensures that the collected data is relevant and covers the various ways in which the model is expected to be used in real-world applications. For example, a team that is working on an Optical Character Recognition (OCR) system can identify typical usage scenarios, such as processing receipts, forms, and documents, then collect and prioritize test data for each scenario, ensuring comprehensive coverage and efficient testing.
- Consulting experts and conducting user studies help in understanding the diverse scenarios in which the ML model operates. This collaborative effort aids in categorizing scenarios and tailoring the data collection process to meet specific needs. This approach helped in curating a more focused and effective test dataset, ensuring that the model is tested across a wide range of real-world applications.

Test Input Generation: In ML testing, the generation of test inputs plays a critical role in identifying vulnerabilities and ensuring the robustness and reliability of ML systems. Research in this area has developed sophisticated methodologies to generate diverse types of test inputs, each targeting different aspects of system vulnerabilities [9].

- Domain-Specific Test Input Synthesis focuses on generating realistic inputs that closely mimic real-world data, divided into adversarial and natural inputs. Adversarial inputs are intentionally perturbed to test the robustness or security of ML systems, potentially exposing flaws by presenting the system with rare or unexpected scenarios. On the other hand, natural inputs are drawn from the typical data distribution seen in practical applications, aiming to ensure the ML system performs well under normal operating conditions. Techniques like DeepXplore leverage white-box testing and neuron coverage to create inputs that uncover differences between DNN models, aiming to simulate as closely as possible real-world data conditions. Other methods involve using generative adversarial networks (GANs) for creating highly realistic data transformations, such as altering weather conditions in driving scenes to test autonomous driving systems, demonstrating the critical role of domain-specific synthesis in ensuring comprehensive testing of ML systems.
- Fuzz and search-based test input generation techniques involve randomly generating data or methodically searching the input space to find cases that cause ML models to fail or behave unexpectedly. These approaches are effective in identifying vulnerabilities, including crashes, memory leaks, or performance issues. For example, TensorFuzz applies a hill-climbing strategy to maximize coverage and discover faults in TensorFlow models, while DLFuzz focuses on generating adversarial examples to test neural network resilience.

- Symbolic execution is utilized to analyze software for potential violations of desired properties, generating inputs that cover different execution paths through the software. When applied to ML testing, symbolic execution can be used to target either the ML model's data or its code, aiming to uncover inputs that lead to errors or unexpected behavior. Techniques like DeepCheck translate a neural network into a symbolic program to find vulnerabilities, such as inputs causing misclassifications.
- Generating synthetic data for testing learning programs involves creating data that mimic real-world distributions or scenarios to evaluate ML models' performance and resilience. This technique is particularly useful for testing models under conditions or with data types that may not be readily available in existing datasets.

Testing ML Model Source Code: Unlike traditional software, where input and output can be precisely defined and expected behaviors are known, ML models involve complex computations and rely on statistical patterns within the data. This makes it difficult to ascertain whether a model is correctly implemented based solely on its output. The intricate nature of models, especially DNNs, which consist of numerous arithmetic and matrix operations, complicates the task of writing test cases that can effectively capture potential issues in model implementation or data handling.

Practices like code reviews and overfitting checks (e.g., training the model on a small, simple dataset to see if it can achieve near-perfect performance) are employed as indirect methods to assess model correctness. However, these do not guarantee the identification of all potential issues, especially subtle bugs that might affect model performance in production scenarios.

4.2 Test Execution

Regression Problem: When updating ML models to new versions, a significant challenge encountered is the regression problem, where the new model, despite passing more test cases than its predecessor, fails on some cases that the older version could handle. This issue is more pronounced in ML systems due to their statistical nature, making it harder than in traditional software to address these setbacks. The problem is critical because it affects users' trust, especially in scenarios where reliability is essential. Users expect the newer versions of systems to maintain or improve upon the previous outputs, and failure to do so can lead to confusion and a loss of confidence in the system. Moreover, the complex and often opaque nature of ML algorithms, particularly DL models, makes identifying and rectifying these regression issues challenging, with no straightforward solutions currently available. This situation highlights the need for new approaches to manage and reduce regression problems in ML systems, an area still open for research and development.

The Oracle Problem: Unlike traditional software, where outcomes can be predicted with certainty, ML algorithms often produce probabilistic results, making it hard to pinpoint errors directly. This issue is tackled through various inventive approaches, such as metamorphic testing, cross-referencing, and employing model evaluation metrics as test oracles. These methods aim to circumvent the traditional need for a predetermined correct outcome, offering innovative ways to assess ML model performance and reliability [9].

Metamorphic relations, rooted in traditional software testing, leverages the relationship between changes in software input and the resultant changes in output across multiple executions to identify bugs without the need for a precise expected outcome. Essentially, if altering an input in a known way leads to an unexpected change in output, it may indicate a flaw in the system under test. In ML testing, this concept is particularly useful because it allows for the testing of models even when the exact output may not be known beforehand.

Cross-referencing compare the outputs of similar applications or different versions/implementations of the same algorithm given identical inputs. This method is particularly useful when dealing with "non-testable" programs, as it helps identify discrepancies that may indicate bugs.

Measurement metrics offer a quantitative basis for designing test oracles in ML systems, focusing on non-functional aspects like robustness, fairness, and interpretability. These metrics don't provide direct answers on whether a system is functioning correctly, but they help in assessing key qualities that could indicate underlying issues.

Test Adequacy: The concept of test adequacy in the realm of both traditional software and ML testing centers on evaluating the capability of tests to uncover faults effectively, thereby offering a measure of confidence in the testing process. In traditional settings, this often involves assessing code coverage and performing mutation testing, measures that aim to ensure that as much of the code as possible is executed during testing to reveal potential defects. However, the unique nature of ML systems, where decision logic is derived from data rather than explicitly coded, necessitates different approaches. For ML models, especially DNNs, coverage criteria have been adapted to consider aspects like neuron activation, which reflects how well the internal structures of the model are tested against various inputs. This includes innovative metrics like neuron coverage, which looks at the proportion of neurons activated by test inputs, and more granular measures that capture specific behaviors and interactions within the model. Additionally, mutation testing in ML focuses on introducing minor changes to the model or data to test the resilience of the ML system and its ability to maintain accuracy despite these perturbations.

Despite these specialized approaches, test adequacy in ML faces challenges due to the opaque nature of ML systems, often described as "black boxes". This complicates the direct application of traditional coverage and mutation criteria, as it's harder to correlate these measures with the actual decision-making logic of the models. To address this, researchers have proposed criteria like surprise adequacy, which assesses the novelty of test inputs relative to the training data, aiming to ensure that the range of potential inputs is thoroughly explored without being redundant. Moreover, rule-based checking offers a structured way to evaluate an ML system's functionality, encompassing not just the model but also the infrastructure and data it relies on, and ensuring the system's reliability over time. These methods underscore the complexity of ML testing and the ongoing efforts to adapt and refine testing strategies to better align with the distinct characteristics of ML systems [9].

4.3 Debugging

Bug Report Analysis: In the field of ML, a thorough examination of bug reports from various ML systems, such as Apache Mahout, Lucene, and OpenNLP, revealed key insights into the nature and impact of bugs in these technologies. It was found that the largest portion of ML bugs, approximately 22.6%, is due to incorrect implementations of algorithms, making these bugs not only the most common but also the most severe and time-consuming to resolve. Further studies, including an analysis of TensorFlow bugs, underscored similar findings, pointing out the main causes of such issues as misuse of APIs, unaligned data tensors, and incorrect model parameters. Additionally, in the context of autonomous driving systems, a significant portion of system disengagements during testing was attributed to errors in ML and decision control, highlighting the paramount importance of addressing these flaws for improving the reliability and safety of ML systems [9].

Debug and Repair: Data resampling is mentioned as a method to enhance model accuracy by including generated test inputs in the training data. Specifically, debugging frameworks like Storm and tfdbg offer unique tools for this purpose: Storm simplifies the debugging process by generating smaller, manageable programs, and tfdbg provides detailed insights into TensorFlow model operations, facilitating effective debugging and model correction. These strategies not only aim to identify and rectify faulty components within ML models, such as "faulty neurons," but also improve overall model reliability and performance by addressing and fixing bugs at various levels of the ML system [9].

4.4 Test Result Analysis

This section presents how practitioners analyze the results of testing ML systems and the challenges they face [9].

Evaluation Metrics Selection: Choosing the right evaluation metrics for ML models is a complex challenge. With ML's broad spectrum, including supervised and unsupervised learning, and the diverse properties of models like accuracy, robustness, and fairness, identifying the perfect metrics is daunting. For instance, although numerous metrics for assessing a model's robustness exist, there's no universally accepted standard, leading to teams often evaluating their models based on ad-hoc, non-standardized methods. This situation is further complicated when considering hierarchical ML systems, where combining metrics to gauge overall system robustness is still an area without clear guidelines. The search for effective evaluation techniques remains a significant open question in the field, one that is crucial not only for the development of reliable ML systems but also for guiding future research towards practical metric design.

Online A/B ML model experiment: Online A/B testing involves comparing two versions of a web page or application (version A and version B) with live traffic to determine which one performs better based on predefined metrics. This method is crucial for assessing the performance of ML models in real-world scenarios but is complicated by privacy regulations that limit direct user input analysis. Instead, metrics based on user behavior are developed to evaluate user satisfaction with ML systems, though this is challenging due to its reliance on user interactions and context. It is important to link offline and online metrics to ensure that improvements in lab settings translate to real-world benefits, and crafting effective online evaluation strategies for ML models can be challenging.

Enhancing Interpretability through Data Slicing: Beyond just looking at numbers to see how well ML models perform, a significant majority of experts dive deeper to understand the "why" behind performance changes. Utilizing data slicing, a technique that emerges as a key practice in analyzing the performance of ML models, allows teams to identify specific areas of strength or weakness. By analyzing performance changes across different segments of data, practitioners can formulate targeted improvements. This technique exemplifies the industry's move towards more nuanced and interpretative approaches to ML system evaluation, aiming to bridge the gap between statistical performance and practical effectiveness. However, despite these advancements, understanding the complexity and interconnectedness of ML systems to fully grasp the reasoning behind test results remains a challenging task.

4.5 Uncertainty Estimator

Uncertainty estimators are tools within neural networks that help determine how reliable a model's predictions are. They work by distinguishing between known (data seen during training) and

unknown (data not seen during training) information, effectively recognizing when the model is faced with unfamiliar data and thus how much trust to place in its outputs. This process of quantifying uncertainty is crucial for developing trustworthy ML systems, especially when deploying these models in safety-critical applications where reliability is paramount. By understanding and quantifying the uncertainty, these systems can better assess their own reliability, offering insights into their limitations and areas where they may need further training or adjustments [16].

The practical importance of uncertainty estimation is underscored by its application across various domains, from medical diagnostics to autonomous driving and quality assurance in manufacturing. In each case, the goal is to enable systems to make informed decisions about their level of confidence in their predictions. This not only helps in enhancing the safety and reliability of automated systems but also provides a mechanism for human operators to understand when to trust the system's outputs and when additional scrutiny is needed. Thus, uncertainty estimators serve as a critical component in bridging the gap between the capabilities of ML models and the practical requirements of real-world applications, ensuring that these systems can be deployed responsibly and effectively.

4.6 Test Properties

ML properties concern the conditions we should care about for ML testing and are usually connected with the behaviour of an ML model after training [19].

Correctness: In ML systems, correctness is basically how accurately these systems perform their tasks. Classic validation techniques, such as hold out cross-validation, k-fold cross-validation, leave-one-out cross-validation, and bootstrapping, are crucial in assessing this correctness. These methodologies involve dividing the dataset in specific ways to isolate training and test data, thereby enabling the evaluation of how well the trained model generalizes to new, unseen data. The choice of validation method plays a crucial role in ensuring the reliability of the ML system, with each approach offering its own advantages in terms of assessing model performance and generalization capabilities. Moreover, the measurement of correctness extends beyond simple accuracy, including a variety of metrics such as precision, recall, and the Area Under the Curve (AUC). These metrics, however, come with their limitations—for instance, accuracy may not differentiate between types of errors made by the model, and precision and recall could be misleading in cases of unbalanced data. The selection of appropriate performance metrics is thus critical and requires careful consideration to accurately reflect the system's correctness.

Model Relevance: Evaluating model relevance is crucial for identifying mismatches between a model and its data, which often show up as either overfitting or underfitting. Cross-validation, while traditional, may not always effectively detect overfitting, especially if the test data does not represent potential unseen data well. Innovative approaches like Perturbed Model Validation (PMV) offer a more comprehensive approach by adding noise to training data and assessing the model's sensitivity to this change, thereby providing a clearer understanding of both overfitting and underfitting compared to traditional methods. Additionally, ongoing data collection post-deployment and the generation of adversarial examples have been proposed as methods to continuously gauge and mitigate overfitting. These methods, alongside leveraging training time as a complexity measure and re-sampling training data, present a multifaceted approach to enhancing model relevance and accuracy, underscoring the dynamic and evolving challenge of fitting models to data accurately without succumbing to overfitting or underfitting.

Robustness: Robustness in ML systems is about how well a system can maintain its performance when faced with unexpected or noisy data. A robust system should correctly handle inputs that deviate from the norm without significant drops in performance. Various methods and metrics have been developed to measure and enhance the robustness of these systems. For instance, tools like DeepFool help identify small changes in input data that can 'fool' a system, highlighting its vulnerabilities. Additionally, researchers have proposed multiple robustness metrics, such as pointwise robustness, adversarial frequency, and adversarial severity, to quantify how small changes to inputs affect a system's outputs.

Security concerns in ML also focus on the system's resilience against intentional manipulations aimed at causing incorrect outputs. Techniques like adversarial input generation are used to test the system's defenses against such manipulations, which is critical for applications in sensitive fields like autonomous driving. The aim is to ensure that ML systems are not only accurate under ideal conditions but also reliable and secure in the face of real-world challenges and potential attacks.

Fairness: Fairness in ML refers to the system's ability to make decisions without discrimination or bias towards any group or individual. Unfairness can arise from several sources, including biased data samples, labeling practices, limited features, disproportionate sample sizes, and the use of proxy features. The field of fairness in ML seeks to identify, measure, and mitigate these biases to ensure equitable treatment across different demographics. Various fairness metrics and definitions have been proposed, such as Fairness Through Unawareness, Group Fairness, Counterfactual Fairness, and Individual Fairness, each with its own approach to assessing and ensuring fairness in ML models.

Ensuring fairness in ML is not only a technical challenge but also a societal imperative to prevent harm and ensure equitable outcomes. Different fairness metrics provide frameworks for evaluating and improving fairness, but no single definition is universally accepted. Techniques like Themis and Aequitas focus on test generation to identify and mitigate unfairness, highlighting the importance of continuous assessment and improvement of ML systems. In addition, Tools and algorithms like RobinHood and fairness-aware programming languages have been developed to support multiple fairness definitions, enabling more equitable ML applications.

Interpretability: Interpretability in ML is crucial for understanding how models make decisions. Currently, the most common method to assess interpretability involves manual evaluation with human participants. This approach can vary from application-specific testing, where real-world scenarios are used, to more general human-based evaluations on simpler tasks, and even to functionally-grounded evaluations that rely on quantitative metrics instead of human input. Models like decision trees and logistic regression have been shown to be more understandable to humans compared to complex models like neural networks. On the other hand, automatic assessment methods are being developed to evaluate model behavior without human intervention, focusing on simpler, inherently interpretable models and using specific metrics and patterns to aid user understanding. These efforts aim to make ML models more transparent, especially in critical areas like medicine, where understanding the basis for predictions is essential.

5 MODEL DEPLOYMENT

Deploying ML models in real-world applications is not just about developing these models but also involves integrating, monitoring, and updating them efficiently in production environments. These tasks present unique challenges, including managing complex software systems over time. There's a field called DevOps that focuses on these types of tasks, and applying its principles to

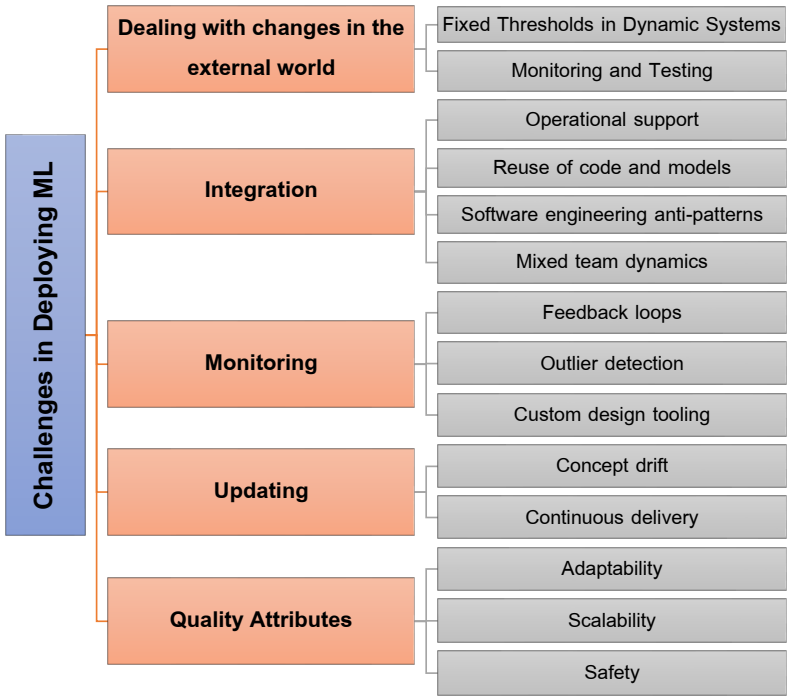

Fig. 4. Challenges in Deploying ML

ML—sometimes referred to as AIOps—can help address some of these challenges. However, ML deployment also faces unique issues like collecting high-quality data and maintaining the models, which requires continuous effort to integrate new data and monitor performance to ensure the models remain effective and relevant over time. Figure 4 illustrates the challenges of deploying ML in a structured manner, which will be discussed in further detail in the following sections.

5.1 Dealing with changes in the external world

Dealing with changes in the external world is crucial for the maintenance of ML systems, given their direct interaction with a dynamic environment. The inherent instability of the external world necessitates ongoing adjustments to these systems, highlighting the challenge of setting and updating decision thresholds. Traditional methods, which often involve manually setting thresholds to balance metrics like precision and recall, become impractical as models evolve. Automated learning of thresholds through evaluation on validation data is suggested as a mitigation strategy to reduce the brittleness of manual updates.

Moreover, comprehensive monitoring and testing are vital to ensure the reliability of ML systems amidst external changes. This includes monitoring for prediction bias to ensure the predicted labels' distribution matches observed labels, establishing action limits for real-world actions to avoid spurious triggers, and closely observing upstream data sources to maintain data integrity. The emphasis on automated alerts and responses is underscored, especially to address time-sensitive changes without solely relying on human intervention, which can be a brittle response mechanism

in rapidly changing environments [15].

5.2 Integration

Integrating a ML model into production involves building the necessary infrastructure and adapting the model so it can be effectively used and supported. This process is crucial and combines elements of both ML and SE, highlighting the importance of code and data reuse to save time, effort, and resources. For instance, Pinterest managed to streamline their systems by creating a universal set of image embeddings used across multiple models, which simplified their deployment process and enhanced task performance. The integration process also reveals common challenges in ML software development, such as managing external data dependencies and avoiding engineering *anti-patterns* like glue code or configuration debt. Collaboration between researchers and SE is key, emphasizing the need for joint ownership of the codebase, shared version control, and active participation in the entire development cycle to ensure successful and efficient product delivery [12].

5.3 Monitoring

Monitoring ML systems involves tracking data quality, model performance, and bias to ensure they operate correctly over time. This process is challenging due to the dynamic nature of data and the complexity of ML models, especially as they can influence their own input data through *feedback loops*. *Outlier detection* is crucial for identifying predictions that deviate significantly from expected outcomes, which can signal problems with the model's generalization ability or calibration. Effective monitoring requires *custom tools and approaches*, as off-the-shelf solutions often don't meet the specific needs of ML systems. This highlights the importance of tailored monitoring strategies to maintain high model performance and data integrity in production environments [12].

5.4 Updating

Updating ML models is crucial to keeping them aligned with the latest data trends, involving techniques like regular retraining and continual learning. A key challenge is *concept drift*, which happens when the data distribution changes, potentially degrading model performance. Addressing concept drift requires careful monitoring and updating of strategies. Additionally, updating models in production involves not just the model itself but also the associated data and code, which complicates the deployment process. This complexity underscores the importance of *continuous delivery* approaches to streamline updates, ensuring models remain effective and trustworthy. Ensuring updates do not disrupt user trust or system compatibility is also critical, highlighting the delicate balance between model accuracy and consistency in behavior post-update [12].
Therefore, another challenge is raised in this area, known as *backward compatibility*, which refers to the ability of a model to maintain its performance and reliability when updated or integrated into larger systems without introducing new errors that were not present in earlier versions. This concept is crucial for ensuring that improvements to a model do not disrupt existing dependencies or expectations in real-world deployments. The challenge arises because updates aimed at enhancing model accuracy can inadvertently decrease backward compatibility, leading to unexpected behavior and reliability issues. For example, an update might improve overall model performance but introduce errors in specific cases that were previously handled correctly, affecting downstream applications or user trust. Studies highlight the need for careful consideration of backward compatibility in the design and update of ML models to prevent unforeseen degradation of system performance and maintain trust with end users [17].

5.5 Quality Attributes

Adaptability: When deploying ML models, the difference in data between the training phase and actual use can cause performance issues due to data distribution changes, known as skew. New data collection and unstable data dependencies might introduce biases if not carefully managed. The challenge is also in maintaining the data extraction pipeline to ensure it applies the same transformations as during training, which can be labor-intensive. Furthermore, when new training data is collected from the model's predictions, it could create a feedback loop that affects the training data for future models, necessitating the isolation of training data from the influence of previously deployed models to prevent bias [11].

Scalability: Deploying ML models in real-world applications involves meeting strict latency requirements and managing large volumes of requests. Some systems handle this by performing much of the computation offline, leading to challenges in quickly adapting to changes. The evolution towards real-time updates and online learning represents a shift in managing scalability, emphasizing the need for efficient and flexible ML serving infrastructure [11].

Safety: In the context of safety, ML deployment in critical domains like autonomous vehicles is approached with caution. Prototypes are built and evaluated in real-life scenarios to develop and test ML components safely. However, there are no reported cases of challenges from commercially deployed ML systems in safety-critical areas, indicating either a lack of deployment in these domains or an absence of documented issues. This gap highlights the need for further research and development to ensure ML systems can be safely integrated into applications where failure could lead to severe consequences [11].

6 CROSS-CUTTING ASPECTS

6.1 Optimization

This section presents groundbreaking research papers that explain innovative optimization methods for ML systems.

Hardware Optimization

Optimization in the context of ML systems, particularly regarding the use of hardware accelerators like GPUs and TPUs, is a critical challenge that influences the performance and efficiency of models. Optimizing ML algorithms for these accelerators is difficult due to their unique architecture, which requires careful consideration of memory systems and parallel processing capabilities. This challenge is exacerbated when attempting to implement new or unconventional algorithms, where existing optimization tools and kernels may not suffice. [2] elaborates on the concept of "monolithic kernels," which are highly optimized, pre-compiled pieces of code for specific operations such as 2D convolution. While these kernels offer high performance for standard computations, they limit flexibility and hinder the adoption of innovative ML ideas that do not fit into the predefined computational paradigms. The optimization process thus becomes a bottleneck, requiring significant manual effort to achieve comparable performance for new algorithms. Moreover, ML framework APIs are inflexible, which makes it difficult to implement custom computational primitives efficiently. The reliance on pre-optimized, monolithic kernels forces researchers to fit new ideas into existing computational models, often at the cost of performance and expressiveness. This situation creates a "rut" for ML systems, where the difficulty of optimization discourages exploration and

innovation in algorithm design, ultimately slowing down the progress in the field.

The development and optimization of Google's Tensor Processing Unit (TPU) exemplify progress in addressing these challenges. The TPU is specifically designed for DL tasks, especially inference operations, and provides an impressive 92 tera-operations per second (TOPS) capability along with a substantial 28 MiB on-chip memory. This design significantly improves neural network computation performance, positioning the TPU to outperform traditional CPUs and GPUs in terms of speed (achieving a 15X–30X increase) and energy efficiency (with a 30X–80X improvement). Such advancements highlight the critical role of hardware optimization in ML, where the TPU serves as a pivotal example of how specialized hardware can dramatically reduce computational time and energy consumption for large-scale neural network applications, thereby alleviating some of the optimization challenges faced by researchers and developers in the field [7].

Efficient Model Training

In [4] an in-depth analysis of optimizing training strategies for large language models (LLMs) within specific computational constraints is presented. It emphasizes a balanced scaling of model size and training data volume to maximize model performance. Contrary to previous strategies that primarily focused on enlarging the model size, this study suggests that both the model size and the number of training tokens should be increased proportionally to utilize the compute resources efficiently. This approach, exemplified by training the Chinchilla model, achieves superior performance on a wide range of tasks compared to other models trained under similar compute budgets, highlighting the importance of strategic resource allocation in model training optimization.

Expanding on the theme of optimization, "Bagpipe" [1] introduces an advanced system designed to optimize the training process of DL-based recommendation models. The core challenge in training such models lies in the handling of billions of parameters, especially from embedding layers, which significantly slow down the training due to the extensive time spent on embedding access and synchronization. Bagpipe addresses these challenges by employing innovative caching and prefetching techniques, which substantially reduce the overhead associated with remote embedding accesses. This optimization is achieved through a unique component called Oracle Cacher, which leverages lookahead algorithms to make optimal cache update decisions and ensures strong consistency against data staleness. Bagpipe is capable of delivering up to 5.6x speed improvements over existing state-of-the-art methods, without compromising on model convergence or reproducibility, marking a significant advancement in the efficiency of training deep recommendation models.

Further broadening the scope of optimization, "ROLLER" [20] is an innovative tensor compiler designed to optimize DL computations by generating efficient kernels in a remarkably short time. Unlike traditional approaches that rely on extensive search algorithms, often requiring hours to optimize a single DL operator, ROLLER employs a novel tile-based construction method. This method, centered around a tile abstraction called "rTile," strategically encapsulates tensor shapes to align with accelerator hardware features, enabling rapid generation of highly efficient kernels. By limiting the search space through alignment with hardware characteristics, ROLLER achieves significant reductions in compilation time—producing kernels in seconds without sacrificing performance. This advancement not only speeds up DL development cycles but also offers new avenues for exploring computational strategies on emerging hardware platforms.

Sustainability

"Green AI" [3] investigates the energy consumption and computational efficiency of DL frameworks, specifically comparing PyTorch and TensorFlow. It highlights a significant difference in the frameworks' energy use and runtime performance across various models. TensorFlow generally shows better performance in training phases, whereas PyTorch excels in inference tasks, indicating that the choice between these frameworks can have substantial implications for both environmental impact and computational efficiency. This analysis underscores the importance of considering energy efficiency in the development and selection of DL frameworks, contributing to the broader goal of making AI more sustainable and accessible.

6.2 End-users' Trust

Building trust with end-users is crucial for the successful adoption of ML solutions, yet it poses significant challenges due to the inherent complexity and lack of transparency of these models. Engaging users early in the development process, particularly in sectors like healthcare or disaster risk management, has proven to be an effective strategy. Projects such as Sepsis Watch have demonstrated that trust can be fostered through strong communication, stakeholder engagement, and clear accountability mechanisms, even in contexts where previous attempts at automation have failed. This approach is supported by the understanding that while model interpretability is valuable, its effectiveness as a trust-building tool is limited. Instead, making development processes transparent, incorporating user feedback, and ensuring open access to software and data are recommended. Additionally, the design of user interfaces plays a critical role in user adoption, with successful applications showing that interfaces tailored to the specific needs and contexts of the end-users can significantly enhance their trust and confidence in ML systems. This multi-faceted approach underscores the importance of not only technological excellence but also of meaningful user engagement and transparency in building end-users' trust in ML solutions [12].

Recognizing the importance of user trust in the deployment of ML applications, it becomes equally important to address the challenges that arise when these systems encounter unexpected situations, such as Out-of-Distribution (OOD) data [18], which can significantly impact the perceived reliability and safety of these technologies. OOD detection in ML involves identifying data samples that differ significantly from the data on which the model was trained. The necessity for OOD detection arises because ML models typically assume that the data they will predict on in the real world will be similar to the data they were trained on. However, in practical applications, this is rarely the case. Data encountered in the real world can often be significantly different due to various factors like novel scenarios not represented in the training data or shifts in the data distribution over time. Detecting such OOD samples is crucial for maintaining the reliability and safety of ML systems. For instance, in autonomous driving, it's vital that the system recognizes when it encounters scenarios it has not seen before and, therefore, might not be able to make safe decisions on. This process helps in avoiding potentially incorrect or dangerous predictions by allowing the system to take precautionary measures, such as alerting a human operator.

7 CONCLUSION

This report explores the challenges and innovative practices associated with deploying ML systems in the industrial environment. It reveals that practitioners face significant challenges at every stage of the ML deployment workflow, including data management, model deployment, and ethical considerations. The report emphasizes the need for the academic community to actively engage with these challenges, advocating for a holistic approach that bridges the gap between theoretical

research and practical applications. The collaborative effort can advance ML deployment practices by leveraging insights from related disciplines and encouraging the sharing of knowledge and experiences from the field.

REFERENCES

[1] Saurabh Agarwal, Chengpo Yan, Ziyi Zhang, and Shivaram Venkataraman. 2023. Bagpipe: Accelerating deep recommendation model training. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 348–363.

[2] Paul Barham and Michael Isard. 2019. Machine learning systems are stuck in a rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 177–183.

[3] Stefanos Georgiou, Maria Kechagia, Tushar Sharma, Federica Sarro, and Ying Zou. 2022. Green ai: Do deep learning frameworks have different costs?. In *Proceedings of the 44th International Conference on Software Engineering*. 1082–1094.

[4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[5] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–16.

[6] Ben Hutchinson, Andrew Smart, Alex Hanna, Emily Denton, Christina Greer, Oddur Kjartansson, Parker Barnes, and Margaret Mitchell. 2021. Towards accountability for machine learning datasets: Practices from software engineering and infrastructure. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 560–575.

[7] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[8] Alexander Lavin, Ciarán M Gilligan-Lee, Alessya Visnjic, Siddha Ganju, Dava Newman, Sujoy Ganguly, Danny Lange, Atílím Güneş Baydin, Amit Sharma, Adam Gibson, et al. 2022. Technology readiness levels for machine learning systems. *Nature Communications* 13, 1 (2022), 6039.

[9] Shuyue Li, Jiaqi Guo, Jian-Guang Lou, Ming Fan, Ting Liu, and Dongmei Zhang. 2022. Testing machine learning systems in industry: an empirical study. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 263–272.

[10] Ximeng Liu, Lehui Xie, Yaopeng Wang, Jian Zou, Jinbo Xiong, Zuobin Ying, and Athanasios V Vasilakos. 2020. Privacy and security issues in deep learning: A survey. *IEEE Access* 9 (2020), 4566–4593.

[11] 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 (2020), 106368.

[12] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. 2022. Challenges in deploying machine learning: a survey of case studies. *Comput. Surveys* 55, 6 (2022), 1–29.

[13] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Record* 47, 2 (2018), 17–28.

[14] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[15] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015).

[16] Joachim Sicking, Maram Akila, Jan David Schneider, Fabian Hüger, Peter Schlicht, Tim Wirtz, and Stefan Wrobel. 2022. Tailored Uncertainty Estimation for Deep Learning Systems. *arXiv preprint arXiv:2204.13963* (2022).

[17] Megha Srivastava, Besmira Nushi, Ece Kamar, Shital Shah, and Eric Horvitz. 2020. An empirical analysis of backward compatibility in machine learning systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3272–3280.

[18] J Yang, K Zhou, Y Li, and Z Liu. 2021. Generalized out-of-distribution detection: A survey. arXiv. *arXiv preprint arXiv:2110.11334* (2021).

[19] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36.

[20] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, et al. 2022. {ROLLER}: Fast and efficient tensor compilation for deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 233–248.