State-of-the-Art and Challenges of Engineering ML- Enabled Software Systems in the Deep Learning Era

GEBREMARIAM ASSRES*, Kristiania University of Applied Sciences, Oslo, Norway
GURU BHANDARI, Kristiania University of Applied Sciences, Oslo, Norway
ANDRII SHALAGINOV, Kristiania University of Applied Sciences, Oslo, Norway
TOR-MORTEN GRONLI, Kristiania University of Applied Sciences, Oslo, Norway
GHEORGHITA GHINEA, Computer Science, Brunel University London, London, United Kingdom of Great Britain and Northern Ireland

Emerging from the software crisis of the 1960s, conventional software systems have vastly improved through Software Engineering (SE) practices. Simultaneously, Artificial Intelligence (AI) endeavors to augment or replace human decision-making. In the contemporary landscape, Machine Learning (ML), a subset of AI, leverages extensive data from diverse sources, fostering the development of ML-enabled (intelligent) software systems. While ML is increasingly utilized in conventional software development, the integration of SE practices in developing ML-enabled systems, especially across typical Software Development Life Cycle (SDLC) phases and methodologies in the post-2010 Deep Learning (DL) era, remains underexplored. Our survey of existing literature unveils insights into current practices, emphasizing the interdisciplinary collaboration challenges of developing ML-enabled software, including data quality, ethics, explainability, continuous monitoring and adaptation, and security. The study underscores the imperative for ongoing research and development with focus on data-driven hypotheses, non-functional requirements, established design principles, ML-first integration, automation, specialized testing, and use of agile methods.

CCS Concepts: • **Software engineering → Software development life cycle**; • **Artificial intelligence → Machine learning**.

Additional Key Words and Phrases: Conventional software, ML-enabled software, ML-powered systems, SDLC phases, Process areas, Software development models

1 Introduction

Driven by the software crisis of the 1960s, the Software Engineering (SE) discipline was coined and enabled the production of high-quality software [142]. SE aims to adopt methodical approaches to software development, thereby achieving success in implementing software projects. In other words, SE is the application of engineering principles to software, as described in the terminology of the IEEE standard glossary [20]. In seminal work, Wirth [142] pointed out that software systems were promised but could not be completed and delivered on time due to high complexity, particularly after the introduction of time-sharing systems. The SE discipline has introduced systematic and quantifiable approaches to software development, operation, maintenance, and retirement, thereby tackling software complexity. In SE, the Software Development Life Cycle (SDLC) provides a structured process

Authors' Contact Information: Gebremariam Assres, Kristiania University of Applied Sciences, Oslo, Norway; e-mail: Gebremariam.Assres@kristiania.no; Guru Bhandari, Kristiania University of Applied Sciences, Oslo, Norway; e-mail: guru.bhandari@kristiania.no; Andrii Shalaginov, Kristiania University of Applied Sciences, Oslo, Norway; e-mail: andrii.shalaginov@kristiania.no; Tor-Morten Gronli, Kristiania University of Applied Sciences, Oslo, Norway; e-mail: tor-morten.gronli@kristiania.no; Gheorghita Ghinea, Computer Science, Brunel University London, London, United Kingdom of Great Britain and Northern Ireland; e-mail: george.ghinea@brunel.ac.uk.

ACM 1557-7341/2025/4-ART
https://doi.org/10.1145/3731597

to produce high-quality software according to prescribed production quality, cost, and time. The SDLC works based on the core phases, including requirements gathering, software design, development, test and integration, deployment, operation, and maintenance [114].

Artificial Intelligence (AI) has also been used to create autonomous systems with an attempt to replace and or augment human decision-making, which eventually led to the development of Machine Learning (ML), as a means of achieving that same AI goals [34]. Although there have been periods called AI winters throughout its history where AI research and development was quiet [86], today, ML, along with the large amounts of data being produced by diverse types of systems like the Internet of Things (IoT), web applications, corporate databases, smartphones, and sensors is a popular subset of AI. It enables computers to generate actionable insights and build ML-enabled (intelligent) software systems based on previous experiences. In ML-enabled systems, modules or functionalities that incorporate ML techniques and algorithms, namely ML components, are introduced to perform tasks that traditionally require human intelligence. The ML component provides partial autonomy to the automated units, evaluates and optimizes processes, and forecasts future trends [84]. The development of ML is a multi-phase process and uses various types of algorithms (and models like neural networks) to support the decision-making process. The phases in ML model development include data collection, data preparation, model selection, training, evaluating, parameter tuning, and deployment [95]. Although ML algorithms initially focused on solving mathematical problems and object recognition [34], nearest neighbor and K-Nearest Neighbor algorithms have been introduced for pattern recognition and deep learning (DL), which imitates the human thinking process, and has known renewed impetus post 2010, in what is widely considered to be the start of the modern DL era, when increased GPU speed enabled the advent of novel convolutional neural network (CNN) architectures such as AlexNet [68]. Lately, ML's advancement on a global scale has been driven by the emergence of Large Language Models (LLM) and generative AI [36, 135]. According to Wang et al. [135], these models have the ability to generate coherent and contextually relevant text, enabling them to perform various tasks, including text completion, text generation, and serving as conversational AI, among others.

The interaction between elements of the SE and ML disciplines is not a novel subject of study, and numerous pieces of literature have discussed their mutual influence. As an illustration, the research highlighted in [84, 89] explored how these fields intersect, particularly for addressing the challenges in software architectural design, which provide high-level descriptions of software components and their interaction. In this context, ML serves as a tool for enhancing the architectural design of conventional software systems. By conventional software, we mean the classical software-based automation of specific tasks like business functions, websites, etc. In this regard, the literature provides valuable perspectives on the utilization of ML-based tools, techniques, and methods to enhance the field of SE and, consequently, to create high-quality software systems. For instance, ML has found application in automating specific phases of the SDLC, such as software testing, as demonstrated in [148]. There is also a need to create architectural styles, patterns, and frameworks to seamlessly incorporate ML components into the design of ML-enabled software systems[89]. This architectural focus on the interaction between elements of the SE and ML disciplines represents only one aspect of the wide spectrum of SE practices. Nevertheless, existing studies have yet to comprehensively examine how SE tools, techniques, and methods are employed in the development of ML-enabled software systems (i.e., systems powered by ML), particularly across typical life cycle phases and methodologies in the DL era (i.e. post 2010). Thus, in the study reported herein, we adopted a holistic view of SE practices, as will be described next.

We have reviewed the state-of-the-art and challenges concerning SE practices in the development of ML-enabled software systems. This study contributes by conducting a thorough review of the existing literature, offering insights into how the SE discipline is practiced in the context of developing ML-enabled software systems, particularly focusing on the typical SDLC phases (hereafter also referred to as SE process areas) and software development methodologies. By analyzing the findings presented in the prior research, this study provides a comprehensive understanding of the current state of knowledge in the field.

The remainder of the paper is organized as follows. Sections 2 and 3 present the background and methodology of the study. Next, the results of the current practices and challenges of engineering ML-enabled software are provided in Section 4, while Section 5 discusses the results. Finally, Section 6 concludes the paper.

2 Background

Software, designed and custom-built, degrades over time and leverages technological frameworks. Thus, SE integrates processes, methods, and tools, emphasizing an organizational commitment to quality standards using principles like TQM, CMMI, Six Sigma, and ISO [31, 51, 71, 108, 121]. The SDLC phases, partition development into manageable activities- requirements specification[131], design[104], development, testing[52], deployment[25], and maintenance[97]- thereby achieving the standards. SE methods offer technical guidelines, addressing defects, schedules, resources, and costs. Examples include waterfall, prototyping, spiral, and agile. SE tools, like Computer Aided Software Engineering (CASE) tools, automate tasks, enhancing productivity and quality through structural or object-oriented paradigms such as diagramming tools, automated testers and code generators [17, 50, 107–109]. In the subsequent subsections, we introduce the core concepts and modern approaches in ML as well as their interaction with SE in the development of ML-enabled software.

2.1 Core Concepts and Applications in Machine Learning

Before delving into the development of ML-enabled systems, it is essential to provide an explanation of what AI entails. The literature indicates that AI is a difficult term to define robustly [30]. However, various authors have made a few historical attempts to define it. For example, one of the most commonly used definitions of AI is stated as "the simulation of human intelligence in computers that are programmed to think like humans and mimic their actions" [110].

AI systems can be designed as rule-based systems or learning-based systems. Rule-based systems (also known as expert systems) are the simplest forms of AI, which are created using a set of rules along with basic data as knowledge representation. These form AI models that mimic the reasoning capability of human experts in solving knowledge-intensive problems [39]. AI-based on computer learning, or ML, generates its models through extensive datasets representing the domain. That is, AI is an umbrella discipline that covers everything related to making machines smarter, while ML refers to the subset of AI that implements models that can self-learn based on algorithms and get smarter over time without human intervention.

2.1.1 Principles and applications of ML. Sarker [115] describes today's digital world as being endowed with data obtained from IoT, cyber security, mobile, business, social media, health applications, etc. ML plays a key role in analyzing these data and developing smart applications.

ML methods are commonly categorized as supervised, unsupervised, semi-supervised, and reinforcement learning in the area. Such models are used to enhance the intelligence and capabilities of applications in various real-world domains, such as cyber security systems, smart cities, healthcare, e-commerce, agriculture, and more [115].

Linear regression, logistic regression, decision tree, support vector machines (SVM), Naive Bayes, neural networks, K-means clustering, and random forest are among the algorithms used in the development of ML models [115]. Examples of common ML applications include traffic alerts, social media, automated language translation, transportation and Commuting, dynamic pricing and product recommendations, virtual personal assistants, self-driving cars, etc.

DL and deep neural networks are also part of ML methods that can intelligently analyze data on a large scale [113, 115]. The major phases of ML development are data collection, data pre-processing, model selection, training the model, model evaluation, parameter tuning, and making predictions.

In the development of ML models, the quality of the resultant model is significantly influenced by the data pre-processing phase. This crucial step involves evaluating and improving the quality of data through operations such as data cleaning, transformation, and reduction. These processes aim to address various issues like missing data, data inconsistency, incorrect formats, and data types, among others, as highlighted by Samek et al. [113]. Despite its importance, the data pre-processing phase often receives inadequate attention in ML development.

2.1.2 The need for large datasets. Data is of paramount importance throughout various phases of the development of ML models. The general consensus is that a larger training dataset contributes to improved model performance. Consequently, substantial data collection from diverse sources, including enterprise applications, websites, emails, IoT devices, smartphones, and sensors, is imperative for ML model development [90, 120, 134]. Samek et al. [113] emphasize the necessity of selecting representative features during training, avoiding sample sizes that are too small or too large. Ensuring model performance involves scrutinizing data through train-test-validate splits and fine-tuning after each training phase.

2.1.3 Quality considerations in ML. Both data and algorithms play critical roles in ensuring the quality of ML-enabled systems in terms of performance, robustness, reliability, fairness, scalability, etc. However, most researchers and practitioners concentrate more on algorithms while undervaluing the impact of data quality. Many domain-specific techniques are used to assess and improve the quality of data stored in relational databases, which necessitates evaluating their suitability in ML. In addition, there are trans-domain and generic dimensions of data quality in the context of ML, including business rules and governance standards for data quality; documented data specifications and integrity maintenance; data consistency, currency, duplication, completeness, provenance, and heterogeneity; data streaming, sampling, dimension reduction, and outliers; feature selection and extraction; data accuracy and bias; and security, namely, confidentiality, privacy, availability and access control [40]. In the case of security, McGraw et al. [79] mentioned the topmost important security risks among several ML-related risks identified in the literature. A description of these risks is provided in Table 1.

The emergence of IoT has also raised several concerns due to smart devices impacting data quality, particularly security and privacy. For example, a study in [16] identified concerns and policy frameworks relating to IoT systems that collect individuals' data through unauthorized surveillance, uncontrolled data generation and use, and inadequate authentication. The study showed that classical privacy policies do not provide adequate protection for the collection and use of individuals' personal data in the context of IoT. Moreover, the diverse data types, data harvesting granularity, and user demographics generated by sensors in IoT systems influence the security and privacy associated with data sharing [2]. Additionally, researchers have investigated IoT quality characteristics relating to commercial voice user interfaces, namely, smart speakers. For example, the study of Pyae and Joelsson [100] investigated the usability, user experiences, and usefulness of Google Home.

2.1.4 ML applications in software development. A study by Meinke and Bennaceur [80] pointed out that ML has been successfully applied in various areas of SE, ranging from software behavior extraction to testing and bug fixing. ML, DL, and LLM applications are foreseeable in software specification extraction, design pattern recognition, code generation, test case generation, bug detection, and learning adaptation strategies in software configuration[32, 80, 136, 137, 145, 150]. ML methods can also be used to predict or estimate software quality, software size, development cost, development effort, reliability, software defect, reusability, release timing, and testability [148].

In regards to the application of ML in software maintenance, Panichella et al. [97] pointed out the following interesting insights for the maintenance and evolution of mobile apps. First of all, ML can provide a high-level taxonomy of categories of sentences contained in the reviews by users that are relevant for maintenance and evolution. Furthermore, it enables the extraction of users' intentions expressed in app store reviews relevant to the maintenance and evolution of apps based on natural language processing (NLP). Similarly, the large amounts

Table 1. Top security risks in Machine Learning

Risk Type	**Characteristic**
Adversarial examples	Adversarial examples are among the popular ML risks where malicious input lead to false prediction.
Data poisoning	In data poisoning, an attacker intentionally manipulates the data in order to compromise the ML system.
Online system manipulation	This is another kind of attack where an attacker can nudge a system in operation (still-learning) through wrong input thereby slowly behaving incorrectly.
Compromised base	A compromised base system may be used in transfer learning, thereby a risk by unanticipated behavior defined by an attacker.
Data confidentiality	These kinds of attacks may extract sensitive and confidential information from ML-enabled systems that used such data during the training.
Data trustworthiness	Lack of data trustworthiness can cause risk due to limitations in the data source like unreliable sensors and lack of data integrity.
Lack of reproducibility	In ML-enabled systems, lack of reproducibility of results due to poor description and reporting can cause risks as a compromise may happen unnoticed.
Overfitting	An ML system may “memorize” its training data set through a lookup table due to overfitting (not generalize to new data) which leads to an adversarial examples attack.
Encoding integrity	Encoding integrity (e.g., metadata) issues can bias a model to solve a categorization problem by overemphasizing the metadata and ignoring the real issue.
Output integrity	Output integrity can cause risk due to unverified output from opaque models where an interposing attacker may hide in plain sight.

of accessible data generated as source code (and other software artifacts) by the software industry can be used to learn patterns and develop productivity tools like NLP-based software code searching, code recommendation, and automatic bug fixing [9]. According to Bader et al. [9], such large amounts of source code are available in GitHub as well as in other proprietary repositories. It also exists in the form of other software artifacts, such as incremental changes between repository code versions, continuous integration tests with outcomes, and developers’ replies on online forums such as Stack Overflow. Abubakar et al. [1] also discussed aspects of the interplay between SE and ML in regard to the estimation of effort and quality in software projects. Furthermore, the authors foresee exploring the possibility of SE-ML fusion in terms of scaling-up operations, tool integration, and performance evaluation. Meinke and Bennaceur [80] also describe a trend towards agile software development to leverage the potential of ML in incremental and exploratory coding.

Search-based SE (SBSE) is another area of application of ML in SE which enables meta-heuristic search techniques to generate adequate software tests evaluated with respect to the fitness function. Harman [46] describes this as an approach to solving SE problems of developing noisy, ill-defined software systems, competing, conflicting, connected, complex, and interactive. In this context, the introduction of ML in SE plays a significant role in realizing the move from an unrealistic utopia of perfection into a more realistic but imperfect software development practice.

Overall, ML in SE offers streamlined processes for tasks like software behavior extraction, testing, and bug fixing [80]. For NLP-based software code searching, ML enhances the precision and speed of code retrieval [9]. It also enhances cost, size, effort, and quality estimation in SE projects, improving planning and decision-making, thereby simplifying complex tasks and contributing to the realization of a realistic software development practice [1, 46, 148]. Moreover, ML aids in predicting software reliability, reusability, testability, and release timing, optimizing resource allocation [148]. However, its application in design pattern recognition and code generation can be intricate, requiring careful modeling and specialized expertise. ML excels in test case generation and bug detection but may lack transparency in decision-making. Data quality and specialized knowledge are essential considerations, highlighting the trade-offs and complexities of ML in SE.

2.2 Engineering Machine Learning-Enhanced Software Systems

In the context of this research, we define ML-enabled software as software augmented with ML components. This type of software leverages ML to carry out tasks that typically demand human intelligence, such as language translation, image recognition, or decision-making. The ML component can be trained with extensive data to execute these tasks with exceptional precision. Consequently, the integration of ML into software enhances its ability to perform intricate tasks swiftly and effectively, surpassing the capabilities of traditional software in isolation.

Engineering ML-enabled software is another dimension of the SE and ML disciplines' interplay. In light of this, various authors claimed that special treatment is needed when developing ML-enabled systems. For instance, according to Martínez-Fernández et al. [78], ML-enabled systems are software with functionalities enabled by at least one ML component. Such components may be used for image recognition, speech recognition, traffic prediction, product recommendations, self-driving vehicles, email spam filtering, malware filtering, virtual personal assistant, and fraud detection. All of these factors lead to the need to pay special consideration to technical, ethical, and social concerns in the engineering of ML-enabled systems. Accordingly, Gasser and Almeida [35] proposed a layered model for ML governance and introduced principles for developing accountable ML algorithms (namely, responsibility, explainability, accuracy, auditability, and fairness), which have a significant social impact.

Amershi et al. [4], in their study, pointed out that there is widespread interest in integrating ML into conventional software, which in turn necessitates a change in the software development process. The authors also mentioned aspects of ML that make it fundamentally different from conventional software development. These aspects include much more complex discovery and management of data, very different skill requirements for model customization and reuse, and components that are more difficult to handle as distinct modules. In a related context, Ozkaya [93] explained those inherently different characteristics of ML-enabled systems, which she described as software-reliant systems that include data and components that implement algorithms mimicking learning and problem-solving- due to their probabilistic nature (as opposed to the deterministic nature conventional software systems). Although they have many commonalities with regard to building, deploying, and sustaining conventional software systems, the author pointed out that systems with ML components can have a high margin of error (due to the uncertainty that often follows predictive algorithms), which makes ML-enabled systems hard to test and verify.

It has also been highlighted that requirements engineering needs a tailored software development process when applied to the development of ML-based complex systems [11]. However, according to Belani et al. [11], there is no process in place specifically tailored to deal with requirements suitable for specifying such software solutions. From the perspective of software quality and testing, Lenarduzzi et al. [72] asserted that ML applications are produced by developers who lack in-depth knowledge regarding SE processes, which resulted in poorly tested and very low-quality ML-enabled software systems.

2.3 Background Summary

This section provides an overview of SE principles and their integration with ML, highlighting their interplay in modern software development. SE encompasses processes, methods, and tools aimed at maintaining software quality. The SDLC partitions development into phases—requirements specification, design, development, testing, deployment, and maintenance—each essential for achieving quality standards and improving productivity. Focused on self-learning algorithms, ML enhances machine intelligence using supervised, unsupervised, semi-supervised, and reinforcement learning methods, facilitating advancements in cybersecurity, healthcare, e-commerce, and more. Techniques such as linear regression, neural networks, and DL underpin ML's ability to process and derive insights from vast datasets. The integration of ML into SE practices has shown considerable promise through task automation, such as software testing and cybersecurity solutions. Additionally, studies have explored adapting SE principles to address the distinct challenges posed by ML-enabled systems, known for their probabilistic ML algorithms. Despite these advancements, the literature lacks a detailed exploration of how SE principles can effectively support the development of ML-enabled software systems, ensuring robustness, reliability, and ethical standards in their deployment. Addressing this gap, our research aims to investigate the seamless integration of SE and ML, thereby contributing to the advancement of both disciplines. The following section will outline the methodology used to review existing literature, aiming to identify current trends and gaps in this field.

3 Methodology

In this study, our objective is to distill the core insights derived from the empirical experiences of researchers regarding the interaction between the fields of SE and ML. Specifically, we focus on exploring the current landscape and challenges in SE practices related to the development of ML-enabled software systems. Our goal is to analyze and amalgamate existing research to gain a deeper understanding of fundamental principles and draw conclusions regarding the layered technology[108], focusing on the SE process areas and software development methodologies. In this section, we present our review guideline, literature selection strategy, and method of analysis.

3.1 The Adopted Review Guideline

A review guideline serves as a fundamental framework that shapes our methodology for structuring the review process. In this section, we have summarized existing review guidelines [59, 63, 92, 105], which are instrumental in ensuring the integrity and reliability of our study.

The guidelines put forth by Keele et al. [59] and Kitchenham and Charters [63] emphasize the critical milestones in the review process, encompassing the definition of objectives, the execution of the review, and the reporting of findings. These comprehensive guidelines outline a series of detailed activities, which include establishing a review protocol, conducting systematic searches, making selection decisions (such as inclusion and exclusion criteria), data extraction, analysis of results, and the subsequent discussion and conclusion. In addition, Okoli and Schabram [92] proposed an extended guideline that incorporates quality appraisal and synthesis as supplementary components.

Furthermore, there is a qualitative (phenomenological) review guideline, as described by Randolph [105] and Creswell and Poth [21], with the specific aim of elucidating the "lived experiences" of individuals in relation to a particular phenomenon. This guideline encompasses a sequence of steps, including bracketing, data collection, identification of meaningful statements, interpretation, and the comprehensive description of the observed phenomena.

For our study, we have chosen to adopt a review guideline that encompasses the definition of objectives, literature searching, selection processes (inclusion criteria), data extraction, analysis of findings, and the subsequent discussion and conclusion. In line with the phenomenological approach, as suggested by Randolph [105], we have deliberately set aside our own personal experiences, biases, and preconceived notions related to the introduction of ML-enabled software as a phenomenon. This approach ensures that our review maintains an objective and unbiased perspective in exploring the subject matter as it appears in the studies we have examined.

3.2 Literature Search and Selection Strategy

In line with our review objective and the adopted review guideline, five reputable digital libraries, SpringerLink, Scopus, ScienceDirect, IEEExplore, and ACM-DL, were chosen to collect the related studies from January 2010. This time frame was selected as it marked the beginning of the modern DL era, prompting the establishment of thousands of AI startups dedicated to DL [26].

In the literature search, we employed several keywords relating to requirements specification, design, development, testing, deployment, maintenance, and development methodologies- in the context of engineering ML-enabled software (see Section 2). Additionally, we employed inclusion criteria as part of our literature search strategy and formulated search queries to perform an advanced search on the digital libraries. The search queries consisted of alternative search terms or synonyms as operands as well as the 'AND' and 'OR' operators. Accordingly, we ran the search queries below and collected journal and conference articles from the chosen digital libraries. The search string is constructed according to the general pillars adopted in this study- "software development phases" AND "integration with ML" AND "software development methodologies".

"software engineering" OR "requirement specification" OR "requirements engineering" OR "software construction" OR "software design" OR "software architecture" OR "software implementation" OR "software testing" OR "software deployment" OR "software maintenance" OR "user support" OR "software release" OR "software analysis" OR "software configuration management" OR "software quality" **AND**

"AI-based" OR "AI-powered" OR "AI-enabled" OR "artificial intelligence-based" OR "artificial intelligence-powered" OR "artificial intelligence-enabled" OR "ML-based" OR "ML-powered" OR "ML-enabled" OR "intelligent software" OR "AI-augmented" or "ML-augmented" OR "AI-infused" OR "ML software" OR "AI software" **AND**

"agile OR scrum OR kanban OR waterfall OR spiral OR "component-based" OR DevOps OR iterative OR lean OR "extreme programming"

Additionally, we formulated the below search query to address the shorter string length requirement of ScienceDirect. In the search string, we used two AND operators on three operands based on variants of terminologies related to "software engineering", "ML-enabled" or "AI-based", and "agile".

"software engineering" AND ("AI-based" OR "AI-powered" OR "AI-enabled" OR "ML-based" OR "intelligent software" OR "AI-infused" OR "AI software") AND "agile"

Table 2 outlines the criteria for selecting and excluding primary literature in the study, ensuring a focus on relevant, high-quality, peer-reviewed work.

Table 2. Inclusion and exclusion criteria for the primary studies

Criteria	Inclusion	Exclusion
Time Frame	Studies published from January 2010 onwards	Studies published before 2010
Source	Journal and conference papers from SpringerLink, Scopus, ScienceDirect, IEEExplore, and ACM-DL	Papers from sources outside these five digital libraries
Type of Publications	Peer-reviewed journal articles and conference and workshop papers	Non-peer-reviewed articles, books, theses, grey literature
Keywords and Scope	Focus on software development phases and integration with ML, software development methodologies (see search string)	Studies not related to ML-enabled software development or methodologies
Language	English-language papers	Papers in languages other than English

Consequently, a total of 412 journal and conference articles were gathered. Figure 1 illustrates the distribution of these acquired articles across various years within each digital library. Subsequently, we applied filters based on title, abstract, and full-text content to focus on papers related to software engineering practices in developing ML-enabled systems. Papers related to the application of ML-based tools in software engineering were excluded, leading to the identification and selection of 40 articles that were deemed pertinent to our research.

In addition to using the above search strings, we have conducted forward and backward snowballing by selecting initial papers guided by our data extraction process, which involved filtering based on title, abstract, and full text. Snowballing served as a validation method for our search, resulting in a total of 26 papers added to our analysis. A summary of the number of queried and selected primary studies from each digital library is provided in Appendix A.

3.3 Analysis of Secondary Studies

Several systematic literature reviews and mapping studies highlight various aspects of SE practices for ML-enabled systems, covering areas such as non-functional requirements, architecture, project management, and software quality assurance.

- **Non-functional requirements in ML-enabled systems.** De Martino and Palomba [24] classify and discuss challenges in managing non-functional requirements (NFRs) in ML-enabled software. The authors highlight key concerns such as fairness, transparency, security, and performance optimization, emphasizing the necessity of automated tools to handle these aspects. The study underscores that ML systems require continuous monitoring and adaptation to ensure compliance with NFRs.
- **Architectural considerations.** Nazir et al. [89] explore architectural challenges and best practices for ML-enabled systems. They identify major design trade-offs, such as balancing model accuracy with computational efficiency, handling uncertainty in ML predictions, and ensuring API consistency across different ML components. The study also stresses the importance of modularizing ML functionalities to improve maintainability and scalability.
- **SE practices for ML.** Nascimento et al. [88] provide a systematic review of SE practices applied to ML software. The authors identified gaps in traditional SE methodologies when applied to ML-enabled systems,

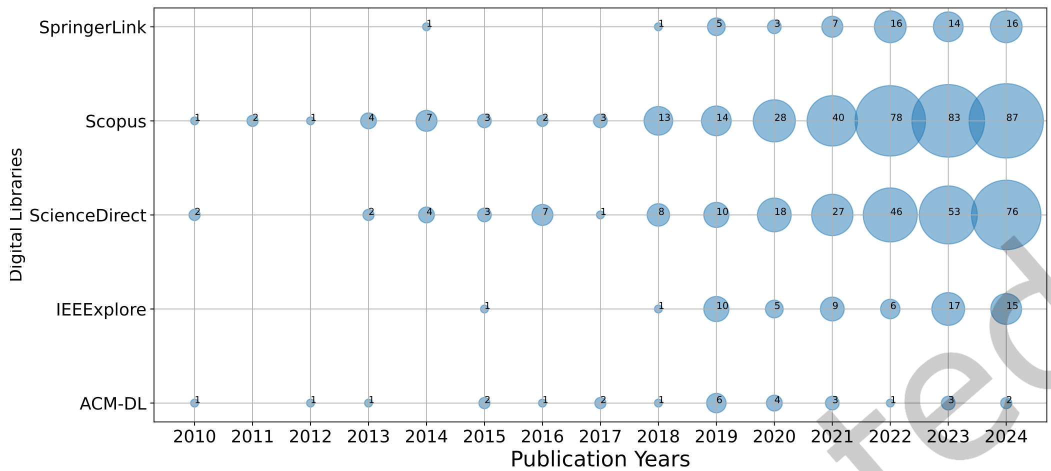

Fig. 1. The yearly distribution of the collected papers in each digital library included in the study.

particularly in requirements engineering, testing, and continuous integration. The study suggests for adapting SE frameworks to better accommodate the iterative and data-driven nature of ML development.

- **Software project management.** Cerdeiral and Santos [17] examine software project management in high-maturity settings, providing insights relevant to ML-enabled systems. The study highlights the need for flexible project management approaches that accommodate the experimental nature of ML development, emphasizing iterative cycles and continuous feedback loops.
- **ML in SE practices.** Wang et al. [137] investigate the role of ML in SE itself, reviewing how ML models are being used to enhance various SE tasks, including defect prediction, code generation, and automated testing. The authors suggested that while ML techniques can improve software quality, they also introduce new challenges related to interpretability and reliability.
- **ML for automated software maintenance.** Zhang et al. [150] focus on the application of LLMs for automated program repair. The findings indicate that LLMs can significantly enhance software maintenance processes, but the authors also highlighted issues such as hallucination, lack of explainability, and the difficulty of integrating ML-driven repair techniques into traditional SE workflows.
- **ML in domain-specific applications.** Antonopoulos et al. [6] conduct a systematic review of ML approaches in energy demand-side response. Although domain-specific, the study provides broader insights into how ML engineering practices must adapt based on industry-specific constraints, data availability, and operational requirements.

Overall, the reviewed secondary studies collectively highlight the complexities and evolving nature of engineering ML-enabled software. While traditional SE practices provide a foundational framework, they often fall short in addressing ML-specific challenges such as data dependencies, evolving model behavior, and NFR compliance. Moreover, there is a strong need for automated tools to streamline NFR management, testing, and continuous integration. Flexible architectural patterns are essential to support modularization, uncertainty management, and scalable deployment of ML models. Interdisciplinary collaboration between ML practitioners and software engineers is crucial to bridging the gap between model development and software system requirements. Additionally, enhanced project management approaches are required to align with the iterative and experimental nature of ML workflows.

3.4 Method of Analysis and Interpretation

In this research, we adopted the chosen articles as our units of analysis rather than conducting direct interviews with individual experts who are affiliated with the domain [105]. Essentially, we relied on existing studies as secondary data sources to elucidate the prevailing engineering practices for developing ML-enabled software within the realm of the interaction between ML and SE.

The chosen studies were subjected to further review aimed at identifying meaningful statements relevant to each area within the SE practices. To achieve this, we gathered empirical assertions presented by the authors regarding the practices and challenges associated with the development of ML-enabled software systems, preserving them verbatim in a spreadsheet. Subsequently, these empirical claims were rephrased to provide clarity and context, as discussed in Section 5. The findings are presented through tables, line charts, donuts and pie charts to offer a visual representation.

4 Results and Analysis

In this section, we delve into the core results of our review concerning the state-of-the-art and challenges encountered in the realm of engineering ML-enabled software. Our aim is to provide insight into the current practices of SE process areas and methodologies in the development of ML-enabled software systems while shedding light on the challenges that researchers, developers, and industry practitioners face. Our analysis not only offers an overview of the field but also paves the way for a deeper understanding of the interplay between SE and ML. Accordingly, results concerning the SE process, each process area, and software development methodologies will be presented next.

4.1 SE Process Areas

Our analysis of the selected studies indicate that research on the engineering practices for developing ML-enabled software has increased in the last decade, as shown in the bubble chart in Figure 1, and we anticipate for this trend to continue growing. Next, we investigated the distribution of the selected studies focusing on each of the typical SDLC phases. The donut chart in Figure 2 illustrates each process area and corresponding percentage distribution in the selected studies.

The analysis includes a citation map graph (Figure 3) that delineates the interconnection of selected studies within the SE process areas- requirements, design, coding, testing, deployment, and maintenance. This graph provides an overview of the citations associated with each process area, revealing additional information on whether a study addresses general concerns pertaining to the process area for ML-enabled software or delves into aspects specific to ML components. The examination of selected studies extends to a detailed exploration, emphasizing the authors' viewpoints on the current practices in implementing the typical SDLC phases in the development of ML-enabled software. Below, we offer a description of the authors' perspectives concerning requirements specification, design, coding, testing, deployment, and maintenance.

4.2 Requirements Specification

The authors of the selected studies have presented diverse perspectives on the prevailing practices in implementing requirements specifications for the development of ML-enabled software, particularly in terms of the integration level between conventional software components and ML components. In this regard, Rahman et al. [103] is among the studies that attempted to reflect on requirement specifications for both components. As per the authors' insights, crafting requirement specifications for ML-enabled applications entails a blend of ML-specific

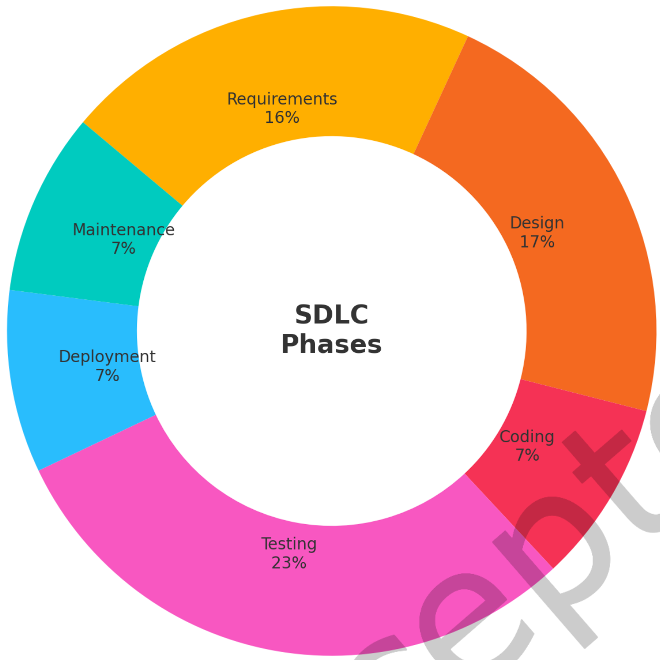

Fig. 2. Percent distribution of the selected studies in SE process areas.

and traditional requirements engineering activities utilized in developing conventional software. They highlight that the specifications for the ML component may undergo frequent changes, posing a challenge in precisely describing the requirements.

Czarnecki [23] also pointed out insights regarding the nature of requirements engineering in the context of software for autonomous vehicles (AV). The functionality of AV needs to be data-driven, which requires expert-assisted and continuous extraction of driving specifications from traffic data. Similarly, Muhammad [85] considers AV in urban environments and presents the importance of specifying human factors such as trust, acceptance, and safety as requirements for the communication between pedestrians and AV. This enables the building of AV with enhanced safety, trust, driving performance, as well as AV-driver interaction.

A review by Martínez-Fernández et al. [78] noted that 60% of their selected studies concentrated specifically on non-functional requirements for ML components. The authors highlighted that these studies predominantly aimed at introducing new ML-specific quality attributes and specification notations to address probabilistic results or ambiguity challenges. Moreover, the review revealed that only a limited number of studies offered a comprehensive perspective on the requirements engineering process for the development of ML-enabled systems.

When ML components are added to conventional software, software developers sustain more challenges in appropriately identifying and comprehending such complex and heterogeneous contexts. In this regard, Wolf and Paine [143] proposed a sense-making theory for conducting requirements specification, thereby making sense of the interaction situation between the requirements specification phases of the development of conventional software and ML-enabled systems.

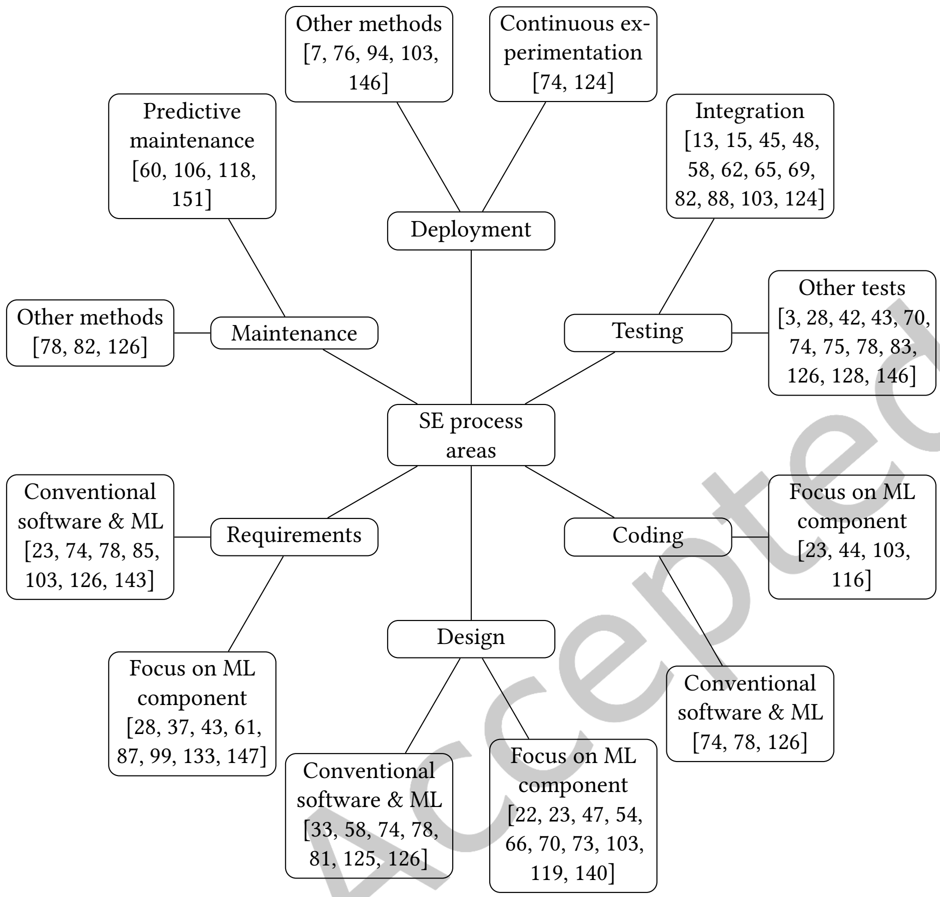

Fig. 3. List of citations under each SE process area.

According to Lu et al. [74], the existing practice of requirements specification often omits or vaguely states the special requirements for building responsible AI. Given the crucial ethical aspect of safety, particularly in ML-enabled systems handling culture-sensitive data, the authors advocate for a more thorough exploration of these requirements. They propose the use of elicitation techniques such as ethical user stories, workshops, interviews, demos, and prototypes. They also suggest categorizing ethical principles into non-functional quality requirements, ensuring verifiability, and maintaining data requirements throughout the SDLC. The practices related to the specification of requirements particular to the ML components are elucidated further in the following studies:

- In a technical briefing on trustworthy AI software, Vakkuri et al. [133] highlighted the incorporation of ethical principles and regulations, such as the General Data Protection Regulation (GDPR), focusing on ML components. The authors outlined commonly featured AI ethics principles, including transparency, justice, fairness, equity, nonmaleficence, responsibility, accountability, privacy, beneficence, freedom, autonomy, trust, sustainability, dignity, and solidarity.

- Habibullah et al. [43] emphasized that ML-reliant systems impose distinct demands on non-functional requirements compared to conventional systems. Traditional requirements like model accuracy are augmented with the addition of explainability.
- Addressing the challenges in planning ML projects due to uncertainty, Nahar et al. [87] proposed mitigation strategies, including incorporating buffer times. They highlighted data security as a non-functional requirement and stressed the benefits of a managerial understanding of SE and ML to align product and model teams toward common goals.
- Dey and Lee [28] underscored safety and robustness as crucial ML requirements, noting the absence of adequate requirements analysis and modeling techniques to handle uncertainty. The authors advocated for explicit requirements specification related to data, ML model, and ML process. Furthermore, they suggested the establishment of quantitative and measurable qualitative targets for explainability, ethical, legal, and robustness aspects of non-functional requirements.

4.3 Design

Similar to the requirements specification, authors portrayed various perspectives concerning the existing practices of performing the design phase in the development of ML-enabled software- conventional software components and ML components. According to a study [126], **73%** of development projects for ML-based systems apply conventional software design approaches, partially or in full, by adjusting to match user needs on the flow of the design process. The authors also added that the use of SE methods in the development of ML-based systems will increase user satisfaction.

In the domain of software design for ML-enabled systems, Meyer and Gruhn [81] highlighted the application of well-established design principles such as separation of concerns, design patterns, and object-oriented and component-oriented development. The authors introduced the concept of concept-based SE, a fusion of design objectives from component-based SE, encompassing productivity and extensibility with ML considerations, particularly focusing on reinforcement learning accuracy. However, Subramonyam et al. [125] argues that the Human-AI interaction prohibits separation of concerns between user experience designers and developers. According to the authors, this is because human needs must shape the design of ML interfaces, the underlying ML sub-components, and the training data.

In related work, Jüngling et al. [58] advocates for the application of design patterns as a means to visualize ML system designs. They exemplify this with a use case involving a passenger counting system, employing a strategy design pattern that integrates rule-based and ML components. Additionally, the authors propose the adoption of a unified modeling language (UML) to facilitate communication of design descriptions among software engineers, ML experts, and knowledge engineers. Furthermore, Lu et al. [74] delves into trustworthiness-by-design, identifying critical factors such as data, algorithm, architecture, and the entire software. They also highlight ongoing efforts in designing user interfaces for Explainable AI (XAI). Broadly, as highlighted in [78], ML-enabled systems' design, development, and operation differ significantly from conventional software systems. Further insights from various authors on existing practices in software design, with a specific emphasis on ML components, are outlined below.

- In their work, Hartikainen et al. [47] delve into human-computer interaction (HCI) design practices within the realm of ML application development. Their focus on HCI for AI (HCAI) underscores critical design constraints such as trustworthiness and usability, alongside key principles including explainability, transparency, ethics, fairness, responsibility, and sustainability. The authors illustrate these concepts through various ML application domains, ranging from customer service chatbots to enterprise resource planning (ERP) systems and IoT solutions.

- The integration of deep neural network models into software architectures, coexisting with classical code, is addressed by Kusmenko et al. [70]. Their methodology automates the ML development process when incorporating neural networks, emphasizing the design of mathematically intensive algorithms to address complex problems without decomposition.
- Czarnecki [23] explores modular and reconfigurable architectures, employing dependability patterns for an automated driving system utilizing a publish-subscribe framework. The author exemplifies this approach using the Robot Operating System (ROS), where components possess message-based interfaces and support easy runtime reconfiguration.
- Discussing challenges, Rahman et al. [103] emphasize the necessity for flexible design in ML-enabled systems to accommodate swift changes in algorithms and frameworks. They note that the performance of ML-enabled systems may degrade over time due to shifts in data patterns, independent of changes in requirements or the presence of bugs. This dynamic nature makes predicting maintenance requirements challenging, highlighting the importance of design flexibility.

4.4 Coding

The coding phase in conventional software involves software integration and the construction of functions, objects, etc. In the context of developing ML-enabled software, coding extends to tasks such as data pre-processing and model training. Authors offer diverse perspectives on existing practices related to coding as presented next.

- For Martínez-Fernández et al. [78], the ML component in ML-enabled software is viewed as embedded ML code or library, serving as a tangible implementation of ML algorithms.
- Lu et al. [74] introduces ethical knowledge graphs as a tool for implementing ethical principles and guidelines (e.g., GDPR) in ML-enabled systems, automatically assessing application programming interface (API) compliance against AI ethics regulations.
- In the study of Rahman et al. [103], the focus is on the ML component, emphasizing that coding frameworks, libraries, and methods for ML applications should align with the requirements of the target platform. Practices such as code reuse, careful framework selection (e.g., scikit-learn, TensorFlow, Keras), and continuous integration of ML models are advocated. This approach ensures implementation choices that consider portability, compatibility, and adaptability to navigate the rapidly evolving hardware-software ecosystem.
- In the context of automated driving systems, Czarnecki [23] underscores the integration of supervised learning with deep neural networks for implementing ML-based perception functions.

4.5 Testing

In the conventional SDLC, testing serves to evaluate and validate the resulting software[53], focusing on aspects such as bug fixing[49, 57], reduction of development costs, and performance improvement. As ML becomes increasingly integrated into software systems, testing methodologies must evolve to address the unique challenges and requirements posed by ML-enabled applications. Next, we present our analysis of secondary studies categorized as overview of various testing strategies, and challenges in the context of ML systems.

- **Testing approaches.** Authors of the selected studies presented various levels of functional and non-functional testing of ML-enabled systems (i.e., acceptance, unit, performance, regression, and scalability testing), as depicted in Table 3. For example, Syahputri et al. [126] compiled testing methods observed in current studies within the agile methodology. Additionally, Gutierrez et al. [42] introduced fuzzy-based testing as an approach to accelerate operational testing, ensuring the integrity of flight software without system interruption. Similarly, other studies highlighted testing methods such as canary testing, an automated quality assurance approach in the DevOps context[3].

- **Model validation.** Testing in ML, often referred to as model validation, involves assessing the performance of an ML model using data that the model has not been exposed to during training[83, 128]. In DevOps, the validation is usually performed before committing the code and running tests locally. Once the model evaluation meets the performance requirement, the ML code needs to be integrated into the system code for production. Furthermore, testing activities for ML-based software components do not only focus on detecting bugs in source code but also on inherent issues that arise from model errors and uncertainty[4]. Thus, automating the testing process is an important strategy in SE, where testing teams create test cases that capture the required behavior of the ML model.
- **Automation and integration testing.** Furthermore, in distributed environments, integration testing is required and performed after ML model testing aimed at validating and verifying the quality of the developed model[28, 70, 103].
n this regard, Steidl et al. [124] discussed testing as part of the CI/CD (continuous integration- continuous delivery) pipeline, which can be performed either manually, semi-automatically, or automatically- on data, data schema, and models.
- **Early testing and user feedback.** Ensuring the functionality of ML-enabled software through early testing in the development process is essential, especially considering the inherent uncertainty in ML [47]. Employing expert evaluation and gathering feedback from end-users in the initial phases facilitates the early detection of model faults during the iterative ML development process [43, 125, 138, 139].
- **Ethical and quality assurance challenges.** Verification and validation testing plays a pivotal role in meeting the requirement specifications of ML-enabled systems, with ethical acceptance testing offering a means to identify and verify ethics-related design flaws in ML-enabled systems [45, 74]. However, testing ML-enabled software is fraught with challenges. The intricate nature of ML-enabled systems poses numerous testing and quality assurance challenges for both ML components and the entire software product or service [38, 41, 43, 87]. Common challenges include the absence of a clear testing strategy, the low priority assigned to model testing, an unclear commitment to system testing, and a lack of transparency in testing processes and results within teams.
- **Unique quality standards.** The inherent uncertainty in ML models demands specialized expertise for the implementation of rigorous testing, particularly for non-functional requirements in ML-enabled systems [43]. The development of test cases for ML-enabled systems requires unique quality standards to account for the uncertainty associated with ML model outputs [78]. Indeed, Rahman et al. [103] highlighted the formidable challenge of testing and rectifying errors in ML applications, exacerbated by the opacity of ML models, which hampers the understanding and explanation of erroneous behavior.

In the context of DL, advanced techniques in testing and debugging are crucial for improving reliability and performance in ML systems. Comprehensive studies focusing on DL bug characteristics [56, 138, 139] and repairing [49, 57, 149] reveal common bug patterns and challenges in DL systems, as presented next.

- Advanced techniques such as *DeepLocalize* for fault localization with DNNs[139], *UMLAUT* for debugging DL programs using program structure[117], and *DeepDiagnosis* for automatically diagnosing faults and recommending actionable fixes in DL programs[138, 149] to include detailed discussions on automated fault diagnosis and the actionable fixes recommended by these systems. These studies emphasize the importance of structural analysis and automated diagnosis for localizing faults in DL models.
- An automated bug debugger system— MODE[77] focuses on debugging by using state differential analysis and strategic input selection to identify and correct anomalies within the model.
- Similarly, *AUTOTRAINER*[152] automates the detection and repair of common training issues in deep neural networks, such as vanishing gradients and incorrect data preprocessing, by implementing solutions like adjusting learning rates and modifying architectures.

- These security-related studies on ML-enabled software also discuss the limitations of the recent advances in software security.

This structured overview of testing methodologies and their implications in ML-enabled software development highlights the importance of adapting existing practices while addressing the unique challenges posed by ML technologies.

Table 3. Authors' insights on ML-enabled software testing practices

Testing Methods	**Characteristics**
Fuzzy testing	Fuzzy testing utilizes random input data to identify vulnerabilities and enhance robustness without interrupting operations[42].
Canary testing	Allows users to assist in a live environment to validate features before full deployment[3].
ML testing/evaluation	Evaluation of ML model, and used for ML optimization[43, 75, 83, 128].
System verification	Verification of the developed system in pre-production environments, semi-automated or automated processes. Formal models and various types of testing[28, 75].
Integration testing of hybrid system	The deployment is followed by real-time monitoring[58].
Integration testing of distributed systems	Testing occurs after testing the ML model. Testing by integrating with the TORCS simulator[70].
Unit testing	Unit testing frameworks (e.g., PyUnit for Python)[103].
ML cross-validation testing	To ensure the statistical relevance of the results. Avoid overfitting and biases[146].
(Semi-)automatic and iterative validation	Testing of data, data schemas, and models in the CI/CD pipeline[82, 124].
Ethical acceptance testing	Define testable acceptance criteria for ethical principles, integrating tests for ML and non-ML component interactions while considering AI quotient and human factors[45, 74].

4.6 Deployment

The deployment phase describes the process of making a software system available for use on a target environment, such as a production server or end-user device[29]. The deployment process can vary depending on the type of software, the target platform, and the project's specific requirements. In ML-enabled software, deployment involves placing a working ML model in an environment where it should do the task as it is intended to do. Our analysis of secondary studies concerning this is presented next.

- Nguyen-Duc and Abrahamsson [91] pointed out that deployment can be considered as a part of the CD/CD pipeline of DevOps. Moreover, it can be achieved by exposing APIs associated with the ML models and using them as standard libraries when developing other ML-enabled solutions [127].

- Lwakatare et al. [75] pointed out that the deployment of ML-enabled software can be performed as a manual, semi-automated, or automated process in pre-production environments. In a related context, research has examined different deployment approaches for engineering ML-enabled systems, elucidating the associated challenges. This information is succinctly encapsulated in the initial section of Table 4.
- Additionally, a survey presented by Alnafessah et al. [3] summarized continuous re-deployment in a production environment via run-time service management for dynamic resource scheduling of micro-services for ML models.
- In DevOps, CI/CD are key enablers to stabilize, optimize, and automate the deployment process of ML models [38, 124]. These facilitate the provision of an automated infrastructure, higher availability, better support, and incident response for the ML system. However, effective automation requires the provision of consistent APIs, thereby avoiding dependencies with other libraries. Thus, CI deals with merging code into the main branch and automating the system's build and testing.
- The other challenge in DevOps is that the development pipeline can change frequently, making it difficult to reproduce the process outside the local environment without the assistance of specialized data and code version control systems (e.g., git, DVC, etc.) [38, 74, 78, 153]. Thus, monitoring the ML model after deployment and testing must take into account the DevOps (MLOps for ML projects) workflow [38, 83, 128]. Therefore, ML deployment needs proper planning, monitoring, and documentation.
- Lu et al. [74] also presented challenges relating to deployment strategies for responsible AI addressing continual learning based on new data, high uncertainty, and other risks. The strategies include a phased deployment of a subset of the ML-enabled software, initially for a certain group of users, thereby reducing ethical risk and homogeneous redundancy.

4.7 Maintenance

Like in conventional software, maintenance, and support in ML-enabled software consist of performance monitoring and horizontal and vertical scaling [3]. In this regard, once the trained ML component is operational in the actual environment, the system should be continuously monitored to detect issues such as performance degradation, compatibility, portability, and scalability problems [75, 103]. However, ML model deployment and performance optimization introduce maintenance challenges due to large datasets and knowledge transfer. Thus, we present our analysis of secondary studies on this topic as follows.

- Yang and Rossi [146] explained open-set recognition as a key building block for judging the fitness of a trained ML model to its production environment while detecting novelty in individual inferences. It also ensures timely and accurate detection of model performance degradation by tracking multiple inferences of the same model.
- Similarly, studies such as in [8, 124] emphasized the importance of getting collective feedback or alerts during runtime, which can be used to trigger the maintenance subsystem.
- Additionally, minor modifications to the ML model structure and data can exert a substantial influence, causing noteworthy shifts in the performance attributes of the ML modules. Consequently, there is a demand for ongoing maintenance, customization, and reuse of the end-to-end pipeline while it is in production, requiring diverse expertise [4, 94].
- The subsequent segment of Table 4 encapsulates the viewpoints of the authors concerning the operation, maintenance, and support within the domain of engineering ML-enabled systems.

4.8 Development Methodologies

Software development methodologies constitute the integral components of layered technology, playing a vital role in the engineering of high-quality software products or services. Among these methodologies, Agile and its

Table 4. Authors' insights on deployment, maintenance, and support of ML-enabled software

Methods	**Description**
Deployment	
Open-set recognition	Checking the fitting of a trained ML model. Identification of overall model degradation [146].
Continuous experimentation	Allows gathering user feedback during run-time [74, 124].
Continuous monitoring and validation	Dynamic, adaptive, and extensible ethical risk assessment. Version-based feedback, and incentives [74, 82].
Non-critical and critical deployment	Cascading deployment of ML components and autonomous ML components [76].
Maintenance, and support	
Collective feedback during runtime	Get feedback from the end-users (Ops) as soon as possible. Monitoring quality requirements in near real-time [8, 124].
Predictive maintenance (PdM)	Establish action possibilities afforded by PdM systems. Implement the actualization process of these affordances focusing on conceptual adaption and constraint mitigation [60].
Tests tracing and verification	Trace the tests verified in any of the previous phases. Support the domain experts and the technicians to identify faulty components [82].

variants (such as SCRUM) stand out for their recognized attributes of flexibility, dynamism, and adaptability to specific circumstances. These quality attributes are achieved through active customer involvement, incremental delivery, a people-focused approach (i.e., the focus on individuals over processes), embracing change as well as prioritizing simplicity [122, 123]. Considering the aforementioned, there is a noticeable inclination towards incorporating traditional software development methodologies, notably agile frameworks, in ML projects [67, 112]. Therefore, we investigated the development patterns, specifically the adoption of lightweight, scalable, and automated (agile-like) methodologies for ML-enabled software projects. It was observed that **57%** of all the selected studies concentrated on the prevailing practices of integrating ML-enabled software development projects with established development methodologies, as delineated below.

A subset of studies [3, 75, 83, 144, 146]tackled the challenges and potential solutions in developing complex systems incorporating ML components. These studies delved into the practices of utilizing DevOps and ML workflow processes concurrently. Other related literature in [8, 23, 38, 82] demonstrated the adoption of newer DevOps-like terminologies such as AIOps, MLOps, and DataOps to integrate ML into traditional DevOps processes. In a second category, inspired by the agile methodology, studies explored contexts like "Agile for ML-based systems" [128], "Agile4ML" [132], and "Agile-like engineering processes" [4] to assimilate the distinctive characteristics of ML-enabled software projects into modern agile frameworks. Additionally, a study by Halme [45] introduced a method to accommodate the unique ethical requirements of ML projects, namely ethical user stories (EUS), within the agile process.

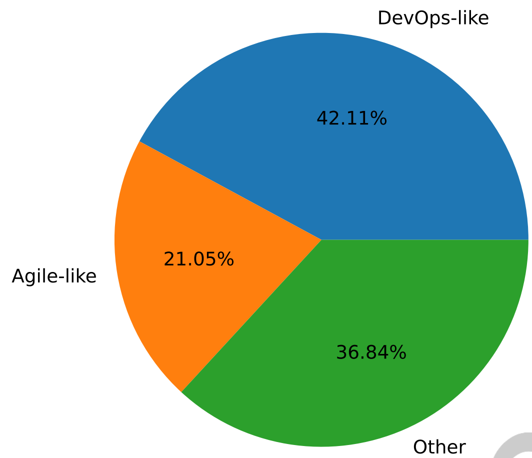

Fig. 4. Percent distribution of selected studies relating to development methodologies.

The third category of studies aimed at envisioning various other software development methodologies, providing general insights into adapting existing methodologies to suit ML-enabled software development projects. This included perspectives such as implementing regulations like GDPR [133], team organization for component-based development [41], and continuous development pipelines for specifying, orchestrating data, training, and integrating (safety-critical) ML-based applications [102, 124]. Moreover, these studies addressed the identification of diverse patterns of approaches in practical ML development projects, projects involving neural networks, and acceptance-oriented continuous experimentation [70, 91, 127].

The distribution of studies among DevOps-like, Agile-like, and other development methodologies is illustrated in Figure 4, where 42.11% of the selected studies concerning software development methodologies focused on DevOps-like methodologies, while Agile-like and other methodologies constituted 21.05% and 36.84%, respectively.

5 Discussion

This section discusses the results (Section 4) and highlights the current practices and challenges in engineering ML-enabled software, focusing on SE process areas and development methodologies.

5.1 Trend Analysis of the Studies

Our analysis, illustrated in Figure 1, underscores an increasing trend in the annual distribution of selected studies, with ScienceDirect and Scopus emerging as dominant repositories in this thematic area. This surge in research activity within ML-enabled software is driven by several contributing factors. There is a growing demand for ML-enabled solutions that aim to enhance efficiency, streamline processes, and extract valuable insights [6, 130]. Consequently, software engineering researchers are delving into the potential of ML to craft intelligent systems, automating routine tasks, optimizing intricate processes, and enhancing overall system performance. Secondarily, the integral role of ML algorithms and data analytics techniques in the field is prompting researchers to explore novel algorithms, models, and methodologies to improve accuracy and efficacy [76]. This interdisciplinary nature of engineering ML-enabled software, often involving collaboration among software engineers, programmers, data scientists, and domain experts [58], fosters knowledge exchange, innovation, and the development of holistic solutions.

The availability of open-source ML tools and frameworks like TensorFlow, PyTorch, and scikit-learn constitutes another driving force, expediting ML development [103]. Researchers can harness these tools to build and test ML-enabled software more efficiently, catalyzing research progress in the field. Furthermore, the imperative to establish industry standards and regulations to ensure safety, reliability, and ethical considerations [133] is steering researchers toward studying the impact of ML and contributing to the formulation of guidelines and best practices.

In general, the escalating trend in studies on ML-enabled software engineering practices is driven by the demand for intelligent solutions, advancements in ML, interdisciplinary collaboration[14], accessibility of ML tools, and evolving industry standards. This trajectory is poised to persist as ML technologies continue to evolve, offering new possibilities in the development of ML-enabled software. Moreover, our detailed exploration of the authors' perspectives has provided valuable insights into the existing practices across each SE process area. This discussion on the selected studies, segmented by the focus on software as conventional, combined software (conventional and ML), or ML alone, forms the basis of our analysis of existing practices in each process area.

5.2 Examining the SE Process Areas

The review results concerning the requirements specification, design, coding, testing, deployment, and maintenance in the development of ML-enabled software are discussed below.

5.2.1 Requirements specification. The analysis of the requirements specification reveals a noteworthy shift in software engineering practices, particularly the addition of new attributes in the domain of non-functional requirements for ML components. These attributes, which include trust, acceptance, safety, transparency, justice, fairness (equity), non-maleficence, responsibility (accountability), privacy, security, beneficence, freedom/autonomy, sustainability, dignity, solidarity, accuracy, and explainability, reflect the evolving landscape of ML-enabled software development. This paradigm shift introduces challenges such as complexity in specifying requirements in adherence to regulations like GDPR [129] and ethical principles inherent to ML, exacerbated by the dynamic nature of requirements, uncertainty, and a lack of effective analysis and modeling techniques. Furthermore, the study underscores the recognition that ML imposes distinct demands on non-functional requirements, measured and defined with respect to the model, data, or the entire system. While the findings highlight a current deficiency in a holistic view of the requirements engineering process for ML-enabled software, it is equally noteworthy that ongoing efforts by researchers and practitioners are actively addressing these challenges. Initiatives include the development of techniques for capturing requirements in the interaction situation between SE and ML practices by leveraging sense-making theory [27, 141]. Additionally, frameworks such as ethical user stories, the incorporation of extra buffer time in project planning to accommodate uncertainty, and the introduction of specification notations capable of handling probabilistic results [54] or ambiguity are indicative of the industry's commitment to overcoming the complexities introduced by the integration of ML components' specifications into conventional software.

5.2.2 Design. On the other hand, our exploration of the design phase shows a prevalent trend wherein established design principles developed for conventional software are also applied to the design of ML components[126]. These principles include separation of concerns, design patterns, and object-oriented or component-oriented approaches. However, ongoing endeavors aim to tailor design artifacts specific to ML, introducing innovations such as visual design patterns, concept-based design, strategy design patterns, and the integration of UML for ML-based systems. Noteworthy contributions extend to the realm of user interface design, particularly geared towards XAI, reflecting a nuanced approach to the unique challenges posed by the integration of ML. Additionally, the existing practices underscore a commitment to defining design constraints and principles for ML, emphasizing trustworthiness and usability, and incorporating vital considerations such as explainability, transparency, ethics, fairness, responsibility, and sustainability. Efforts are noticeable in setting architectural design patterns with objectives for seamlessly integrating ML components into classical code, promoting modularity, runtime reconfigurability, and ensuring dependability through message-based interfaces. Furthermore, initiatives addressing the flexibility required to accommodate rapid changes in algorithms and frameworks and proactively managing performance degradation due to evolving data patterns are evident. Yet, despite these strides, the analysis reveals a notable gap, namely the absence of generic design frameworks, architecture styles, and patterns that comprehensively address the unique quality attributes inherent in the development of ML-enabled software.

5.2.3 Coding. In the coding phase, our analysis recognizes that practitioners perceive the ML component in ML-enabled software like an embedded code or library, embodying concrete implementation of ML algorithms[78]. A noteworthy ongoing effort within this field involves the implementation of ethical principles in ML, ensuring that APIs are automatically scrutinized for compliance with regulations governing AI ethics before consumption. This includes the integration of perception functions utilizing deep neural networks. Analogous to established practices in conventional software development, ML developers showcase a commitment to select coding frameworks, libraries, and methods tailored to the nuances of ML software. This ensures the resultant software product or service aligns seamlessly with the requirements of the target platform. Moreover, a recognizable trend in ML coding practices involves the embrace of code reuse [23] strategies, model re-engineering [101], and the adoption of continuous integration methodologies. These approaches are instrumental in navigating the swiftly evolving landscape of ML, fostering adaptability and responsiveness. However, despite these commendable practices, a clear gap remains - the absence of a comprehensive framework that seamlessly integrates ML and conventional software into the cohesive entity, ML-enabled software.

5.2.4 Testing. In ML, testing exposes code bugs, assesses data quality, validates models, and confronts uncertainties prior to code commitment. Our analysis indicates that the practices in ML testing draw upon the existing conventional testing methods, encompassing acceptance, unit, performance, regression, scalability, and integration testing, often seamlessly integrated into CI/CD pipelines[38, 124]. Noteworthy ongoing efforts in this space involve the development and application of specialized ML-centric testing methods, exemplified by fuzzy testing, a dynamic approach performed while the ML system is in operation, and canary testing[3], an automated mechanism for quality assurance within DevOps workflows. Similarly, techniques like fault localization, automated debugging and other metrics are proposed for testing DL- and LLM- enabled software systems[18, 77, 138, 149, 152]. However, ML-enabled software's complex and heterogeneous nature introduces unique testing challenges. The opacity of ML models poses difficulties in achieving explainability, complicating the testing of the entire ML-enabled system. Moreover, the lack of clear test processes, explicit requirements for model development, and robust strategies for system-wide testing further compound the testing landscape. Additionally, the creation of test cases tailored for ML-enabled software necessitates the establishment of quality standards capable of accommodating the inherent uncertainties associated with system outputs.

5.2.5 Deployment. In the deployment phase, our results show an inclination towards adopting deployment practices analogous to those employed in conventional software. This includes deploying ML models as part of the CI/CD pipeline within the DevOps paradigm[38, 124]. Noteworthy practices also involve exposing APIs as standard libraries and employing continuous runtime redeployment for dynamic resource scheduling of microservices. However, deployment in the ML context is not without its challenges. Principal among these challenges is the demand for a high degree of automation in target infrastructure, ensuring availability, providing robust support, establishing effective incident response mechanisms, and maintaining consistency in API provision. Reproducing processes in deployment environments proves challenging, particularly in the face of frequent changes in the development pipeline. Additionally, developing deployment strategies for responsible AI, which incorporates continuous learning and navigates high uncertainty, emerges as a particularly complex task.

5.2.6 Maintenance. Our analysis indicates that the operational phase of ML-enabled software, similar to conventional software, necessitates continuous monitoring to identify defects, encompassing performance degradation and bug detection. Notably, the performance characteristics of the ML component can be significantly altered by minor changes in data or model architecture. Maintenance and support for ML-enabled software are ongoing processes, particularly considering that changes in requirements may necessitate scaling. To address these challenges, current practices include the implementation of open-set recognition for detecting performance degradation in the ML component within its production environment[146]. This approach facilitates the timely initiation of maintenance and support measures. However, our study underscores the heightened complexity of maintaining ML-enabled software, primarily attributed to the large volumes of associated datasets. Moreover, the ML system may need to collect alerts concerning runtime errors, triggering automated maintenance. Consequently, the maintenance of ML-enabled software during its operational phase demands diverse expertise for end-to-end pipeline management.

5.3 Development Methodologies

Our result shows that the selected studies are predominantly focused on software development methodologies. Particularly, the studies portrayed the prevalent adoption of agile frameworks and their variants in ML projects, revealing pivotal trends in the engineering practices of ML-enabled software (see Section 4). This inclination towards established methodologies aligns with the agile principles of flexibility, adaptability, and iterative development, deemed beneficial in the dynamic and evolving landscape of ML[67, 112]. The extensive exploration of DevOps-like terminologies, such as AIOps and MLOps, emphasizes the recognition of ML workflows within broader operational processes. Additionally, the integration of AI ethics through practices like EUS highlights an understanding of ethical dimensions in ML projects within agile methodologies. While fostering adaptability, the prevalence of these methodologies also raises questions about the extent to which they capture the unique challenges and characteristics of ML-enabled software development.

5.4 Discussion Summary

Overall, our results indicate a growing trend in research within the field (see Figure 1), highlighting state of the art, challenges, and best practices- presented next.

5.4.1 State of the art. There is a significant shift towards prioritizing non-functional requirements and the use of automated tools to handle the requirements[24] in the development of ML-enabled software, emphasizing attributes such as trust, transparency, fairness, responsibility, and explainability. Design practices for ML components often leverage established principles developed for conventional software, such as separation of concerns and object-oriented approaches. In coding, ML components are treated as embedded code or libraries, embodying specific ML algorithms. Developers use established practices, frameworks, and methods tailored to ML, including automatic scrutiny of APIs for compliance with AI ethics, code reuse, and continuous integration. ML-enabled software testing integrates traditional and ML-centric approaches, such as fuzzy testing and canary testing, in the CI/CD pipelines. Additionally, methods like fault localization, automated debugging, and various evaluation metrics are suggested for testing DL- and LLM-enabled software systems. Deployment practices adopt CI/CD pipelines akin to conventional software, focusing on API and continuous redeployment. Maintenance practices like open-set recognition are adopted for timely detection of performance degradation due to changing requirements. Development methodologies often align ML projects with agile frameworks, demonstrating adaptability in the dynamic ML landscape.

5.4.2 Challenges. Despite these established practices, several challenges persist. The dynamic nature and demand for unique non-functional requirements such as adherence to regulations (e.g., GDPR) and ethical considerations intrinsic to ML pose challenges[129]. Ongoing efforts seek to introduce innovative design artifacts for ML, such as visual and strategy design patterns, to address distinctive design challenges. The complexity and opaque nature (lack of explainability) of ML models, along with ambiguous test processes, unclear requirements, and the absence of robust system-wide testing strategies, pose challenges for testing ML-enabled software. Deployment faces challenges like high automation needs, ensuring availability, robust support, maintaining API consistency, difficulties in reproducing processes due to frequent changes, developing strategies for responsible AI, and dealing with continuous learning and uncertainty. Maintenance of ML-enabled software is complex due to large datasets, and the requirements for diverse expertise for automated maintenance triggered by runtime errors and comprehensive end-to-end pipeline management introduce more challenges. There is a lower emphasis on innovative methodologies specific to ML integration, underscoring the need for a nuanced approach that adapts existing methodologies and explores innovative strategies. The results highlight that testing receives the most attention among the phases, while deployment and maintenance phases are comparatively underrepresented. However, the deployment and maintenance of ML models should also be given significant emphasis due to the challenges associated with data management, learning, verification, ethics, end-user trust, legal considerations, and security [96].

5.4.3 General Insights and Best Practices. Based on current SE practices in ML-enabled software development, the following best practices are essential for establishing SE tools, techniques and methods as well as for future research in the field:

- **Requirements should begin with hypothesizing potential outcomes from data**, refining them through experimentation. Additionally, strive to align ML performance metrics with business objectives and metrics [37].
- **Prioritize non-functional requirements** such as trust, transparency, fairness, and explainability to ensure ethical and responsible ML component.
- **Leverage established design principles** like separation of concerns, treating ML components as embedded libraries for better modularity. In addition, monitoring performance degradation and handling high-volume data are key ML design considerations, requiring robust architectural patterns.
- **Integrate ML with a two-step process**, first combining ML sub-components, then integrating ML with non-ML system components. Defined interfaces and evolving models require continuous integration support [103].
- **Utilize automated tools** for AI ethics compliance, code reuse, and integrate CI/CD pipelines to streamline development.
- **Implement specialized testing approaches**, including fuzzy and canary testing, alongside automated debugging and performance evaluation metrics. Moreover, automated regression testing and test case prioritization are essential for ML-enabled systems, demanding advanced tools and techniques.
- **Adopt agile frameworks** to foster adaptability and enable continuous integration in dynamic ML component environments.

In general, existing practices in engineering ML-enabled software are often perceived as a fusion of practices from conventional software and ML components, with insufficient recognition of the nuanced interplay between the two. Practitioners do not fully acknowledge the unique characteristics of ML-enabled software, viewing it as a mere collection of separate entities—conventional software and ML. This often results in loose integration, with a focus on developing interfaces to facilitate interaction with the ML component. Thus, there is a need for specialized approaches to ensure seamless integration and delivery of desired quality and functionality across both conventional software and ML components. As ML continues to drive automation across various industries and applications, there will be a growing need to automate various tasks using ML-enabled software.

Appendix B provides an overview of the current practices, challenges, and implications associated with the various process areas and development methodologies in engineering ML-enabled software systems.

5.5 Limitations of the study

Our approach in this study is more like the State-of-the-Art review method [10], concentrating on the latest research in the SE practices for developing ML-enabled software. Thus, insufficient rigor in performing systematic literature review may introduce a potential validity threat (including issues related to internal, external, and construct validity [5]) by potentially limiting the comprehensiveness and replicability of findings. Such common threats to validity in SE may include selection bias, data extraction inconsistencies, and publication bias in the selected studies [64]. However, our study still provides valuable insights based on a structured and thorough analysis of relevant literature.

When gathering related studies, we considered those published after 2010 due to the paradigm shift towards DL, which significantly influenced the proliferation of ML-enabled software systems. However, this criterion may exclude earlier foundational work, potentially limiting completeness, though it enhances relevance by focusing on DL-driven advancements in the larger field of ML.

Our paper selection process for snowballing, based on focus rather than using concrete relevance statistical data, may have also introduced bias. However, aligning with life cycle phases in SE practices ensures contextual validity, while future work could enhance rigor by incorporating quantitative selection criteria.

6 Conclusion

This review paper explores research in engineering ML-enabled software, highlighting state of the art, challenges, best practices, and future research directions. The results indicate a growing trend of research in the field, driven by demand, advancements, collaboration, and evolving standards. In addition, there is emphasis on special non-functional requirements for ML-enabled software and the use of automated approaches to handle them. The findings also highlighted that the development of ML-enabled software integrates both conventional and ML-specific development practices with key challenges being the dynamic nature of ML, opaque models, and complex maintenance requirements, underscoring the need for specialized integration approaches. The replication package of this review study is included on GitHub [12]. Our insights include the need to begin with data-driven hypotheses, prioritize non-functional requirements, apply established design principles, integrate the ML component first, automate, implement specialized testing, and adopt agile methods. Future research should address potential limitations in this study, such as potential biases in literature selection, attrition, and outcome reporting. It is recommended that the review process be rerun with varied contexts, including incorporating more studies from other digital libraries. Additionally, further research is necessary to investigate how the degree of ML integration affects the development process and the quality attributes of ML-enabled (augmented) software. This also includes a thorough analysis of each phase of the SDLC. The engineering of DL- and LLM-enabled software also requires thorough investigation. As the field evolves, challenges such as data quality, ethics, explainability, adaptation, security, legal issues, sustainability, and governance will emerge. Hence, integrating ML, DL, and LLM into existing systems will require careful design, highlighting the need for interdisciplinary collaboration and ongoing research.

References

[1] Hamza Abubakar, Mohammad S Obaidat, Aaryan Gupta, Pronaya Bhattacharya, and Sudeep Tanwar. 2020. Interplay of Machine Learning and Software Engineering for Quality Estimations. In *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*. IEEE, 1–6.

[2] Akash Aggarwal, Waqar Asif, Habibul Azam, Milan Markovic, Muttukrishnan Rajarajan, and Peter Edwards. 2019. User Privacy Risk Analysis for the Internet of Things. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 259–264.

[3] Ahmad Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, and Antonio Filieri. 2021. Quality-Aware DevOps Research: Where Do We Stand? *IEEE access : practical innovations, open solutions* 9 (2021), 44476–44489. doi:10.1109/ACCESS.2021.3064867

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 291–300. doi:10.1109/ICSE-SEIP.2019.00042

[5] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and software technology* 106 (2019), 201–230.

[6] Ioannis Antonopoulos, Valentin Robu, Benoit Couraud, Desen Kirli, Sonam Norbu, Aristides Kiprakis, David Flynn, Sergio Elizondo-Gonzalez, and Steve Wattam. 2020. Artificial intelligence and machine learning approaches to energy demand-side response: A systematic review. *Renewable and Sustainable Energy Reviews* 130 (2020), 109899.

[7] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*. IEEE, 50–59.

[8] Claudia Ayala, Besim Bilalli, Cristina Gómez, and Silverio Martínez-Fernández. 2022. DOGO4ML: Development, Operation and Data Governance for ML-based Software Systems. In *CEUR Workshop Proceedings*, Vol. 3144.

[9] Johannes Bader, Sonia Seohyun Kim, Frank Sifei Luan, Satish Chandra, and Erik Meijer. 2021. AI in Software Engineering at Facebook. *IEEE Software* 38, 4 (2021), 52–61.

[10] Erin S Barry, Jerusalem Merkebu, and Lara Varpio. 2022. State-of-the-art literature review methodology: A six-step approach for knowledge synthesis. *Perspectives on Medical Education* 11, 5 (2022), 281–288.

[11] Hrvoje Belani, Marin Vukovic, and Željka Car. 2019. Requirements Engineering Challenges in Building AI-based Complex Systems. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. IEEE, 252–255.

[12] Guru Bhandari and Gebremariam Assres. 2024. State-of-the-art and Challenges of Engineering ML-enabled Software Systems. https://github.com/SmartSecLab/ML-enabled-software-literature-review.

[13] Houssem Ben Braiek and Foutse Khomh. 2020. On testing machine learning programs. *Journal of Systems and Software* 164 (2020), 110542.

[14] Gabriel Busquim, Hugo Villamizar, Maria Julia Lima, and Marcos Kalinowski. 2024. On the Interaction Between Software Engineers and Data Scientists When Building Machine Learning-Enabled Systems. In *International Conference on Software Quality*. Springer, 55–75.

[15] Junming Cao, Bihuan Chen, Longjie Hu, Jie Gao, Kaifeng Huang, and Xin Peng. 2023. Understanding the Complexity and Its Impact on Testing in ML-Enabled Systems. *arXiv preprint arXiv:2301.03837* (2023).

[16] Xavier Caron, Rachelle Bosua, Sean B Maynard, and Atif Ahmad. 2016. The Internet of Things (IoT) and Its Impact on Individual Privacy: An Australian Perspective. *Computer Law & Security Review* 32, 1 (2016), 4–15.

[17] Cristina T Cerdeiral and Gleison Santos. 2019. Software Project Management in High Maturity: A Systematic Literature Mapping. *Journal of Systems and Software* 148 (2019), 56–87.

[18] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.

[19] Ricardo Colomo-Palacios. 2019. Towards a Software engineering framework for the design, construction and deployment of machine learning-based solutions in digitalization processes. In *The International Research & Innovation Forum*. Springer, 343–349.

[20] IEEE Computer Society. Software Engineering Technical Committee. 1983. *IEEE Standard Glossary of Software Engineering Terminology*. Vol. 729. IEEE.

[21] John W Creswell and Cheryl N Poth. 2016. *Qualitative Inquiry and Research Design: Choosing among Five Approaches*. Sage publications.

[22] Pablo Cruz, Gustavo Ulloa, Daniel San Martin, and Alejandro Veloz. 2023. Software Architecture Evaluation of a Machine Learning Enabled System: A Case Study. In *2023 42nd IEEE International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 1–8.

[23] Krzysztof Czarnecki. 2019. Software Engineering for Automated Vehicles: Addressing the Needs of Cars That Run on Software and Data. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19)*. IEEE Press, Montreal, Quebec, Canada, 6–8. doi:10.1109/ICSE-Companion.2019.00024

[24] Vincenzo De Martino and Fabio Palomba. 2023. Classification, Challenges, and Automated Approaches to Handle Non-Functional Requirements in ML-Enabled Systems: A Systematic Literature Review. *arXiv preprint arXiv:2311.17483* (2023).

[25] Alan Dearle. 2007. Software Deployment, Past, Present and Future. In *Future of Software Engineering (FOSE'07)*. IEEE, 269–284.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[27] Brenda Dervin. 1998. Sense-making theory and practice: An overview of user interests in knowledge seeking and use. *Journal of knowledge management* (1998).

[28] Sangeeta Dey and Seok-Won Lee. 2021. Multilayered Review of Safety Approaches for Machine Learning-Based Systems in the Days of AI. *Journal of Systems and Software* 176 (2021). doi:10.1016/j.jss.2021.110941

[29] Ionut-Catalin Donca, Ovidiu Petru Stan, Marius Misaros, Dan Gota, and Liviu Miclea. 2022. Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors* 22, 12 (2022), 4637.

[30] Wolfgang Ertel. 2018. *Introduction to Artificial Intelligence*. Springer.

[31] John Estdale and Elli Georgiadou. 2018. Applying the ISO/IEC 25010 Quality Models to Software Product. In *European Conference on Software Process Improvement*. Springer, 492–503.

[32] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[33] S. Feldmann, S. Rösch, D. Schütz, and B. Vogel-Heuser. 2013. Model-Driven Engineering and Semantic Technologies for the Design of Cyber-Physical Systems. *IFAC Proceedings Volumes* 46, 7 (may 2013), 210–215. doi:10.3182/20130522-3-BR-4036.00050

[34] Keith D Foote. 2022. The History of Machine Learning and Its Convergent Trajectory towards AI. *Machine Learning and the City: Applications in Architecture and Urban Design* (2022), 129–142.

[35] Urs Gasser and Virgilio AF Almeida. 2017. A Layered Model for AI Governance. *IEEE Internet Computing* 21, 6 (2017), 58–62.

[36] A Shaji George and AS Hovan George. 2023. A review of ChatGPT AI's impact on several business sectors. *Partners Universal International Innovation Journal* 1, 1 (2023), 9–23.

[37] Görkem Giray. 2021. A Software Engineering Perspective on Engineering Machine Learning Systems: State of the Art and Challenges. *Journal of Systems and Software* 180 (Oct. 2021), 111031. doi:10.1016/j.jss.2021.111031

[38] Tuomas Granlund, Vlad Stirbu, and Tommi Mikkonen. 2021. Towards Regulatory-Compliant MLOps: Oravizio's Journey from a Machine Learning Experiment to a Deployed Certified Medical Product. *SN Computer Science* 2, 5 (2021). doi:10.1007/s42979-021-00726-1

[39] Crina Grosan and Ajith Abraham. 2011. Rule-Based Expert Systems. In *Intelligent Systems*. Springer, 149–185.

[40] Venkat Gudivada, Amy Apon, and Junhua Ding. 2017. Data Quality Considerations for Big Data and Machine Learning: Going beyond Data Cleaning and Transformations. *International Journal on Advances in Software* 10, 1 (2017), 1–20.

[41] Rajeev K. Gupta, B. Balaji, V. Mekanathan, and J. Ferose Khan. 2020. Challenges in Scaling AI-powered Distributed Software Product: A Case Study of a Healthcare Organization. In *Proceedings - 2020 ACM/IEEE 15th International Conference on Global Software Engineering, ICGSE 2020*. 6–10. doi:10.1145/3372787.3389300

[42] Tamara Gutierrez, Alexandre Bergel, Carlos E. Gonzalez, Camilo J. Rojas, and Marcos A. Diaz. 2021. Systematic Fuzz Testing Techniques on a Nanosatellite Flight Software for Agile Mission Development. *IEEE access : practical innovations, open solutions* 9 (2021), 114008–114021. doi:10.1109/ACCESS.2021.3104283

[43] Khan Mohammad Habibullah, Gregory Gay, and Jennifer Horkoff. 2023. Non-Functional Requirements for Machine Learning: Understanding Current Use and Challenges among Practitioners. *Requirements Engineering* (2023). doi:10.1007/s00766-022-00395-3

[44] Gaétan Hains, Arvid Jakobsson, and Youry Khmelevsky. 2018. Towards formal methods and software engineering for deep learning: security, safety and productivity for dl systems development. In *2018 Annual IEEE international systems conference (syscon)*. IEEE, 1–5.

[45] Erika Halme. 2022. Ethical Tools, Methods and Principles in Software Engineering and Development: Case Ethical User Stories. In *Product-Focused Software Process Improvement*, Davide Taibi, Marco Kuhrmann, Tommi Mikkonen, Jil Klünder, and Pekka Abrahamsson (Eds.). Vol. 13709. Springer International Publishing, Cham, 631–637. doi:10.1007/978-3-031-21388-5_48

[46] Mark Harman. 2012. The Role of Artificial Intelligence in Software Engineering. In *2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE)*. IEEE, 1–6.

[47] Maria Hartikainen, Kaisa Väänänen, Anu Lehtiö, Saara Ala-Luopa, and Thomas Olsson. 2022. Human-Centered AI Design in Reality: A Study of Developer Companies' Practices A Study of Developer Companies' Practices. In *ACM International Conference Proceeding Series*. doi:10.1145/3546155.3546677

[48] Alaa Houerbi, Chadha Siala, Alexis Tucker, Dhia Elhaq Rzig, and Foyzul Hassan. 2024. Empirical Analysis on CI/CD Pipeline Evolution in Machine Learning Projects. *arXiv preprint arXiv:2403.12199* (2024).

[49] Kai Huang, Su Yang, Hongyu Sun, Chengyi Sun, Xuejun Li, and Yuqing Zhang. 2022. Repairing Security Vulnerabilities Using Pre-trained Programming Language Models. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 111–116. doi:10.1109/DSN-W54100.2022.00027 https://ieeexplore.ieee.org/document/9833850.

[50] Watts S Humphrey. 1988. The Software Engineering Process: Definition and Scope. In *Proceedings of the 4th International Software Process Workshop on Representing and Enacting the Software Process*. 82–83.

[51] Watts S Humphrey. 1995. *A Discipline for Software Engineering*. Pearson Education India.

[52] Timo Hynninen, Jussi Kasurinen, Antti Knutas, and Ossi Taipale. 2018. Software Testing: Survey of the Industry Practices. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 1449–1454.

[53] IBM. 2023. What Is Software Testing and How Does It Work? | IBM. https://www.ibm.com/topics/software-testing.

[54] Vladislav Indykov. 2024. Component-based Approach to Software Engineering of Machine Learning-enabled Systems. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 250–252.

[55] Fuyuki Ishikawa and Nobukazu Yoshioka. 2019. How do engineers perceive difficulties in engineering of machine-learning systems?-questionnaire survey. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. IEEE, 2–9.

[56] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A Comprehensive Study on Deep Learning Bug Characteristics. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 510–520. doi:10.1145/3338906.3338955 https://dl.acm.org/doi/10.1145/3338906.3338955.

[57] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing Deep Neural Networks: Fix Patterns and Challenges. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1135–1146. doi:10.1145/3377811.3380378 https://dl.acm.org/doi/10.1145/3377811.3380378.

[58] Stephan Jüngling, Martin Peraic, and Cheng Zhu. 2022. Using the Strategy Design Pattern for Hybrid AI System Design. In *CEUR Workshop Proceedings*, Vol. 3121.

[59] Staffs Keele et al. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Technical report, ver. 2.3 ebse technical report. ebse.

[60] Robert Keller, Alexander Stohr, Gilbert Fridgen, Jannik Lockl, and Alexander Rieger. 2019. Affordance-Experimentation-Actualization Theory in Artificial Intelligence Research - A Predictive Maintenance Story. In *40th International Conference on Information Systems, ICIS 2019*.

[61] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. 2018. Software engineering for machine-learning applications: The road ahead. *IEEE Software* 35, 5 (2018), 81–84.

[62] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2018. Data scientists in software teams. In *Proceedings of the 40th International Conference on Software Engineering*. ACM.

[63] Barbara Kitchenham and Stuart Charters. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. *Keele University and Durham University Joint Report* 2 (jan 2007).

[64] Barbara Kitchenham, Stuart Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering version 2.3. *Engineering* 45, 4ve (2007), 1051.

[65] Michael Kläs and Anna Maria Vollmer. 2018. Uncertainty in machine learning applications: A practice-driven classification of uncertainty. In *Computer Safety, Reliability, and Security: SAFECOMP 2018 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås, Sweden, September 18, 2018, Proceedings 37*. Springer, 431–438.

[66] Holger Klus, Christoph Knieke, Andreas Rausch, and Stefan Wittek. 2023. Software Engineering Meets Artificial Intelligence. *Electronic Communications of the EASST* 82 (2023).

[67] Iva Krasteva and Sylvia Ilieva. 2020. Adopting Agile Software Development methodologies in big data projects–a systematic literature review of experience reports. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2028–2033.

[68] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[69] Fumihiro Kumeno. 2019. Software engineering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13, 4 (2019), 463–476.

[70] Evgeny Kusmenko, Svetlana Pavlitskaya, Bernhard Rumpe, and Sebastian Stüber. 2019. On the Engineering of AI-Powered Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. 126–133. doi:10.1109/ASEW.2019.00042

[71] Ming-Chang Lee and To Chang. 2006. Applying TQM, CMM and ISO 9001 in Knowledge Management for Software Development Process Improvement. *International Journal of Services and Standards* 2, 1 (2006), 101–115.

[72] Valentina Lenarduzzi, Francesco Lomio, Sergio Moreschini, Davide Taibi, and Damian Andrew Tamburri. 2021. Software Quality for Ai: Where We Are Now?. In *International Conference on Software Quality*. Springer, 43–53.

[73] Grace A Lewis, Ipek Ozkaya, and Xiwei Xu. 2021. Software Architecture Challenges for ML Systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 634–638.

[74] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, and Zhenchang Xing. 2022. Towards a Roadmap on Software Engineering for Responsible AI. In *Proceedings - 1st International Conference on AI Engineering - Software Engineering for AI, CAIN 2022*. 101–112. doi:10.1145/3522664.3528607

[75] Lucy Ellen Lwakatare, Ivica Crnkovic, and Jan Bosch. 2020. DevOps for AI – Challenges in Development of AI-enabled Applications. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 1–6. doi:10.23919/SoftCOM50211.

2020.9238323

[76] Lucy Ellen Lwakatare, Aiswarya Raj, Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2019. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. In *Agile Processes in Software Engineering and Extreme Programming*, Philippe Kruchten, Steven Fraser, and François Coallier (Eds.). Vol. 355. Springer International Publishing, Cham, 227–243. doi:10.1007/978-3-030-19034-7_14

[77] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 175–186. doi:10.1145/3236024.3236082 https://dl.acm.org/doi/10.1145/3236024.3236082.

[78] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software Engineering for AI-Based Systems: A Survey. *ACM Transactions on Software Engineering and Methodology* 31, 2 (2022). doi:10.1145/3487043

[79] Gary McGraw, Richie Bonett, Victor Shepardson, and Harold Figueroa. 2020. The Top 10 Risks of Machine Learning Security. *Computer* 53, 6 (2020), 57–61.

[80] Karl Meinke and Amel Bennaceur. 2018. Machine Learning for Software Engineering: Models, Methods, and Applications. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE, 548–549.

[81] Ole Meyer and Volker Gruhn. 2019. Towards Concept Based Software Engineering for Intelligent Agents. In *Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '19)*. IEEE Press, Montreal, Quebec, Canada, 42–48. doi:10.1109/RAISE.2019.00015

[82] Iori Mitzutani, Ganesh Ramanathan, and Simon Mayer. 2021. Semantic Data Integration with DevOps to Support Engineering Process of Intelligent Building Automation Systems. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '21)*. Association for Computing Machinery, New York, NY, USA, 294–297. doi:10.1145/3486611.3492413

[83] Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi. 2022. MLOps for Evolvable AI Intensive Software Systems. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1293–1294. doi:10.1109/SANER53432.2022.00155

[84] Henry Muccini and Karthik Vaidhyanathan. 2021. Software Architecture for Ml-Based Systems: What Exists and What Lies Ahead. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 121–128.

[85] Amna Pir Muhammad. 2021. Methods and Guidelines for Incorporating Human Factors Requirements in Automated Vehicles Development *(CEUR Workshop Proceedings, Vol. 2857)*.

[86] Nikesh Muthukrishnan, Farhad Maleki, Katie Ovens, Caroline Reinhold, Behzad Forghani, and Reza Forghani. 2020. Brief history of artificial intelligence. *Neuroimaging Clinics* 30, 4 (2020), 393–399.

[87] Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kastner. 2022. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. In *Proceedings - International Conference on Software Engineering*, Vol. 2022-May. 413–425. doi:10.1145/3510003.3510209

[88] Elizamary Nascimento, Anh Nguyen-Duc, Ingrid Sundbø, and Tayana Conte. 2020. Software engineering for artificial intelligence and machine learning software: A systematic literature review. *arXiv preprint arXiv:2011.03751* (2020).

[89] Roger Nazir, Alessio Bucaioni, and Patrizio Pelliccione. 2024. Architecting ML-enabled systems: Challenges, best practices, and design decisions. *Journal of Systems and Software* 207 (2024), 111860.

[90] Americo Talarico Neto, Renata Pontin M Fortes, and Adalberto G da Silva Filho. 2008. Multimodal Interfaces Design Issues: The Fusion of Well-Designed Voice and Graphical User Interfaces. In *Proceedings of the 26th Annual ACM International Conference on Design of Communication*. 277–278.

[91] Anh Nguyen-Duc and Pekka Abrahamsson. 2020. Continuous Experimentation on Artificial Intelligence Software: A Research Agenda. In *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1513–1516. doi:10.1145/3368089.3417039

[92] Chitu Okoli and Kira Schabram. 2010. A Guide to Conducting a Systematic Literature Review of Information Systems Research. *SSRN Electronic Journal* (2010). doi:10.2139/ssrn.1954824

[93] Ipek Ozkaya. 2020. What Is Really Different in Engineering AI-enabled Systems? *IEEE Software* 37, 4 (2020), 3–6.

[94] Andrei Paleyes, Christian Cabrera, and Neil D. Lawrence. 2022. An Empirical Evaluation of Flow Based Programming in the Machine Learning Deployment Context. In *Proceedings - 1st International Conference on AI Engineering - Software Engineering for AI, CAIN 2022*. 54–64. doi:10.1145/3522664.3528601

[95] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. 2020. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Computing Surveys (CSUR)* (2020).

[96] 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.

[97] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How Can i Improve My App? Classifying User Reviews for Software Maintenance and Evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 281–290.
[98] Derek Partridge and Yorick Wilks. 1987. Does AI have a methodology which is different from software engineering? *Artificial intelligence review* 1, 2 (1987), 111–120.
[99] Zhongyi Pei, Lin Liu, Chen Wang, and Jianmin Wang. 2022. Requirements engineering for machine learning: A review and reflection. In *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*. IEEE, 166–175.
[100] Aung Pyae and Tapani N Joelsson. 2018. Investigating the Usability and User Experiences of Voice User Interface: A Case of Google Home Smart Speaker. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*. 127–131.
[101] Binhang Qi, Hailong Sun, Xiang Gao, Hongyu Zhang, Zhaotian Li, and Xudong Liu. 2023. Reusing deep neural network models through model re-engineering. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 983–994.
[102] Martin Rabe, Stefan Milz, and Patrick Mader. 2021. Development Methodologies for Safety Critical Machine Learning Applications in the Automotive Domain: A Survey *(IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops)*. 129–141. doi:10.1109/CVPRW53098.2021.00023
[103] Md Saidur Rahman, Foutse Khomh, Emilio Rivera, Yann-Gaël Guéhéneuc, and Bernd Lehnert. 2022. Challenges in Machine Learning Application Development: An Industrial Experience Report. In *2022 IEEE/ACM 1st International Workshop on Software Engineering for Responsible Artificial Intelligence (SE4RAI)*. IEEE, 21–28.
[104] Paul Ralph and Yair Wand. 2009. A Proposal for a Formal Definition of the Design Concept. In *Design Requirements Engineering: A Ten-Year Perspective*. Springer, 103–136.
[105] Justus Randolph. 2009. A Guide to Writing the Dissertation Literature Review. *Practical Assessment, Research, and Evaluation* 14, 1 (2009), 13.
[106] Gilberto Recupito, Fabiano Pecorelli, Gemma Catolino, Valentina Lenarduzzi, Davide Taibi, Dario Di Nucci, and Fabio Palomba. 2024. Code and Architectural Debt in Artificial Intelligence-Enabled Systems: On the Prevalence, Severity, Impact, and Management Strategies. *Severity, Impact, and Management Strategies* (2024).
[107] Steven P Reiss. 1996. Software Tools and Environments. *ACM Computing Surveys (CSUR)* 28, 1 (1996), 281–284.
[108] S Pressman Roger and R Maxin Bruce. 2015. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Education.
[109] Nayan B Ruparelia. 2010. Software Development Lifecycle Models. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 8–13.
[110] Stuart J Russell and Peter Norvig. 2010. *Artificial intelligence a modern approach.* London.
[111] Dhia Elhaq Rzig, Foyzul Hassan, and Marouane Kessentini. 2022. An empirical study on ML DevOps adoption trends, efforts, and benefits analysis. *Information and Software Technology* 152 (2022), 107037.
[112] Jeffrey Saltz and Alex Suthrland. 2019. SKI: An Agile Framework for Data Science. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 3468–3476.
[113] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. 2021. Explaining Deep Neural Networks and beyond: A Review of Methods and Applications. *Proc. IEEE* 109, 3 (2021), 247–278.
[114] T Saravanan, Sumit Jha, Gautam Sabharwal, and Shubham Narayan. 2020. Comparative Analysis of Software Life Cycle Models. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE, 906–909.
[115] Iqbal H Sarker. 2021. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science* 2, 3 (2021), 1–21.
[116] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2015. On challenges in machine learning model management. (2015).
[117] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. 2021. UMLAUT: Debugging Deep Learning Programs Using Program Structure and Model Behavior. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3411764.3445538 https://dl.acm.org/doi/10.1145/3411764.3445538.
[118] 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).
[119] Yorick Sens, Henriette Knopp, Sven Peldszus, and Thorsten Berger. 2024. A Large-Scale Study of Model Integration in ML-Enabled Software Systems. *arXiv preprint arXiv:2408.06226* (2024).
[120] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2018. A Survey on Sensor-Based Threats to Internet-of-Things (Iot) Devices and Applications. *arXiv preprint arXiv:1802.02041* (2018). arXiv:1802.02041
[121] Brijendra Singh and Suresh Prasad Kannojia. 2013. A Review on Software Quality Models. In *2013 International Conference on Communication Systems and Network Technologies*. IEEE, 801–806.
[122] Ian Sommerville. 2011. *Software Engineering, 9/E.* Pearson Education India.

[123] Apoorva Srivastava, Sukriti Bhardwaj, and Shipra Saraswat. 2017. SCRUM Model for Agile Methodology. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 864–869.

[124] Monika Steidl, Michael Felderer, and Rudolf Ramler. 2023. The Pipeline for the Continuous Development of Artificial Intelligence Models—Current State of Research and Practice. *Journal of Systems and Software* (jan 2023), 111615. doi:10.1016/j.jss.2023.111615

[125] Hariharan Subramonyam, Jane Im, Colleen Seifert, and Eytan Adar. 2022. Solving Separation-of-Concerns Problems in Collaborative Design of Human-AI Systems through Leaky Abstractions. In *Conference on Human Factors in Computing Systems - Proceedings*. doi:10.1145/3491102.3517537

[126] Irdina Wanda Syahputri, Ridi Ferdiana, and Sri Suning Kusumawardani. 2020. Does System Based on Artificial Intelligence Need Software Engineering Method? Systematic Review. In *2020 Fifth International Conference on Informatics and Computing (ICIC)*. 1–6. doi:10.1109/ICIC50835.2020.9288582

[127] Hironori Takeuchi, Kota Imazaki, Noriyoshi Kuno, Takuo Doi, and Yosuke Motohashi. 2022. Constructing Reusable Knowledge for Machine Learning Projects Based on Project Practices. *Intelligent Decision Technologies* 16, 4 (2022), 725–735. doi:10.3233/IDT-220252

[128] Hironori Takeuchi, Haruhiko Kaiya, Hiroyuki Nakagawa, and Shinpei Ogata. 2021. Reference Model for Agile Development of Machine Learning-Based Service Systems. In *2021 28th Asia-Pacific Software Engineering Conference Workshops (APSEC Workshops)*. 17–20. doi:10.1109/APSECW53869.2021.00014

[129] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and YiKe Guo. 2021. Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Computers & Security* 110 (2021), 102402.

[130] Zaib Ullah, Fadi Al-Turjman, Leonardo Mostarda, and Roberto Gagliardi. 2020. Applications of artificial intelligence and machine learning in smart cities. *Computer Communications* 154 (2020), 313–323.

[131] Tousif ur Rehman, Muhammad Naeem Ahmed Khan, and Naveed Riaz. 2013. Analysis of Requirement Engineering Processes, Tools/Techniques and Methodologies. *International Journal of Information Technology and Computer Science (IJITCS)* 5, 3 (2013), 40.

[132] Karthik Vaidhyanathan, Anish Chandran, Henry Muccini, and Regi Roy. 2022. Agile4MLS—Leveraging Agile Practices for Developing Machine Learning-Enabled Systems: An Industrial Experience. *IEEE Software* 39, 6 (nov 2022), 43–50. doi:10.1109/MS.2022.3195432

[133] Ville Vakkuri, Kai-Kristian Kemell, and Pekka Abrahamsson. 2021. Technical Briefing: Hands-on Session on the Development of Trustworthy AI Software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 332–333. doi:10.1109/ICSE-Companion52605.2021.00142

[134] Jayneel Vora, Sudeep Tanwar, Sudhanshu Tyagi, Neeraj Kumar, and Joel JPC Rodrigues. 2017. Home-Based Exercise System for Patients Using IoT Enabled Smart Speaker. In *2017 IEEE 19th International Conference on E-Health Networking, Applications and Services (Healthcom)*. IEEE, 1–6.

[135] Fei-Yue Wang, Qinghai Miao, Xuan Li, Xingxia Wang, and Yilun Lin. 2023. What does ChatGPT say: The DAO from algorithmic intelligence to linguistic intelligence. *IEEE/CAA Journal of Automatica Sinica* 10, 3 (2023), 575–579.

[136] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* (2024).

[137] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. 2022. Machine/deep learning for software engineering: A systematic literature review. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1188–1231.

[138] Mohammad Wardat, Breno Dantas Cruz, Wei Le, and Hridesh Rajan. 2022. DeepDiagnosis: Automatically Diagnosing Faults and Recommending Actionable Fixes in Deep Learning Programs. In *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 561–572. doi:10.1145/3510003.3510071 https://dl.acm.org/doi/10.1145/3510003.3510071.

[139] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*. IEEE Press, Madrid, Spain, 251–262. doi:10.1109/ICSE43902.2021.00034 https://dl.acm.org/doi/10.1109/ICSE43902.2021.00034.

[140] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 49–495.

[141] Karl E Weick. 1995. *Sensemaking in organizations*. Vol. 3. Sage.

[142] Niklaus Wirth. 2008. A Brief History of Software Engineering. *IEEE Annals of the History of Computing* 30, 3 (2008), 32–39.

[143] Christine T. Wolf and Drew Paine. 2020. Sensemaking Practices in the Everyday Work of AI/ML Software Engineering *(Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020)*. 86–92. doi:10.1145/3387940.3391496

[144] Jie JW Wu. 2024. An Exploratory Study of V-Model in Building ML-Enabled Software: A Systems Engineering Perspective. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 30–40.

[145] HanXiang Xu, ShenAo Wang, Ningke Li, Yanjie Zhao, Kai Chen, Kailong Wang, Yang Liu, Ting Yu, and HaoYu Wang. 2024. Large language models for cyber security: A systematic literature review. *arXiv preprint arXiv:2405.04760* (2024).

[146] Lixuan Yang and Dario Rossi. 2021. Quality Monitoring and Assessment of Deployed Deep Learning Models for Network AIOps. *IEEE Network* 35, 6 (nov 2021), 84–90. doi:10.1109/MNET.001.2100227

[147] LI Yishu, Jacky Keung, Kwabena Ebo Bennin, Xiaoxue Ma, Yangyang Huang, and Jingyu Zhang. 2023. Towards Requirements Engineering Activities for Machine Learning-Enabled FinTech Applications. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 121–130.

[148] Du Zhang and Jeffrey JP Tsai. 2003. Machine Learning and Software Engineering. *Software Quality Journal* 11, 2 (2003), 87–119.

[149] Hao Zhang and W.K. Chan. 2019. Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 376–387. doi:10.1109/ASE.2019.00043 https://ieeexplore.ieee.org/document/8952197.

[150] Quanjun Zhang, Chunrong Fang, Yang Xie, YuXiang Ma, Weisong Sun, and Yun Yang Zhenyu Chen. 2024. A Systematic Literature Review on Large Language Models for Automated Program Repair. *arXiv preprint arXiv:2405.01466* (2024).

[151] Xufan Zhang, Yilin Yang, Yang Feng, and Zhenyu Chen. 2019. Software engineering practice in the development of deep learning applications. *arXiv preprint arXiv:1910.03156* (2019).

[152] Xiaoyu Zhang, Juan Zhai, Shiqing Ma, and Chao Shen. 2021. AUTOTRAINER: An Automatic DNN Training Problem Detection and Repair System. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 359–371. doi:10.1109/ICSE43902.2021.00043 https://ieeexplore.ieee.org/document/9402077.

[153] Eduardo Zimelewicz, Marcos Kalinowski, Daniel Mendez, Görkem Giray, Antonio Pedro Santos Alves, Niklas Lavesson, Kelly Azevedo, Hugo Villamizar, Tatiana Escovedo, Helio Lopes, et al. 2024. ML-Enabled Systems Model Deployment and Monitoring: Status Quo and Problems. In *International Conference on Software Quality*. Springer, 112–131.

Appendices

A Summary of queried and selected primary studies

Search String	Digital Library	Queried Studies	Selected Studies
"software engineering" OR "requirement specification" OR "requirements engineering" OR "software construction" OR "software design" OR "software architecture" OR "software implementation" OR "software testing" OR "software deployment" OR "software maintenance" OR "user support" OR "software release" OR "software analysis" OR "software configuration management" OR "software quality" *AND* *"AI-based" OR "AI-powered" OR "AI-enabled" OR "artificial intelligence-based" OR "artificial intelligence-powered" OR "artificial intelligence-enabled" OR "ML-based" OR "ML-powered" OR "ML-enabled" OR "intelligent software" OR "AI-augmented" or "ML-augmented" OR "AI-infused" OR "ML software" OR "AI software"* *AND* *"agile OR scrum OR kanban OR waterfall OR spiral OR "component-based" OR DevOps OR iterative OR lean OR "extreme programming"*	Scopus SpringerLink IEEExplore ACM-DL	196 33 32 23	22 2 12 2
"software engineering" AND ("AI-based" OR "AI-powered" OR "AI-enabled" OR "ML-based" OR "intelligent software" OR "AI-infused" OR "AI software") AND "agile"	ScienceDirect	128	2
Using forward and backward snowballing			26
Total		438	66

B Summary of general insights and best practices

Received 22 November 2023; revised 11 March 2025; accepted 1 April 2025

State of the art & Challenges	General Insights & Best Practices
Requirements Specification	
There is a shift towards non-functional requirements [99, 147] such as data quality, trust, transparency, fairness, safety, and explainability[78]. Efforts are ongoing to capture requirements amid regulatory and ethical complexities. Specifying ML requirements is not straightforward due to its dynamic and uncertain nature[87] and lack of robust analysis techniques. Such requirements are generated inductively from training data which makes it challenging to test and verify [61].	New techniques are essential for effectively capturing requirements and ensuring compliance with regulations like GDPR[129]. However, best practices include clear prioritization and documentation of non-functional requirements like bias assessment tools, fairness and performance metrics (aligning with business objectives[37]), data lineage, and regulatory compliance as well as emphasis for involving stakeholders[22]. Overall, there is a trend towards proposing ML specific guidelines and processes[99].
Design	
Established design principles for conventional software are being applied to ML[126]. Innovations like visual and strategy patterns focus on usability and ethical considerations. Thus, ML models integration requires quality data and explainability[66], and it is often ad hoc with limited architectural patterns available[119, 140]. Has challenges like integrating ML-specific design artifacts and managing rapid algorithmic changes. Properly embedding ML models in systems so that they can be easily maintained or reused is far from trivial[119]. Additionally, there is architecture challenge for addressing monitorability, and co-architecting[73].	There is a need for enhancing modularity, adaptability in design, addressing unique ML software quality attributes, and evaluating architectures for ML-enabled software, considering data abstraction[140], stakeholder knowledge and diverse perspectives [22, 119]. Thus, tailoring traditional component-based approaches[54] for the ML components, and integrating ethical guidelines, usability, visual design patterns for consistency, and strategy patterns for flexible use of algorithms are among the best practices. The resulting complex architectural decision can also be addressed using intelligent automated tools[66].
Coding	
ML components are treated as embedded code[78], implementing specific algorithms with code reuse and continuous integration. There is a lack of comprehensive frameworks for integrating ML code with conventional software[44]. Other challenges include multi-language code base, and challenges in backwards compatibility of trained models[116].	Demands fostering adaptability, responsiveness through code reuse and integration for ML. Thus, best practices include writing modular, reusable code libraries for the ML components, implementing version control, and source code documentation and code completion[78]. The evolving ML components require support for continuous integration[103].

Testing	
Sources of uncertainty to ML components useful in testing are scope compliance, data quality, and model fit[65]. Traditional testing methods are used integrated with ML-centric approaches like fuzzy and canary testing[3]. Ensuring testing aspects like explainability and system-wide strategies for ML models is challenging. Additional challenges include safety, security, verification[69], data quality[62, 88], and integrating ML components into larger systems[15].	There is a requirement for developing robust testing frameworks to address opacity in ML models, ensuring compliance with ethical standards. Thus far, employing fuzzy and canary testing for robustness and gradual rollout, and assessing ethical and fairness requirements through detection tools, evaluation metrics, and use of standardized testing frameworks[48] are among the best practices.
Deployment	
Current deployment practices mirror conventional software, utilizing DevOps CI/CD pipelines for continuous deployment[38, 124]. Challenges in performing automated deployment of production ready models[7], maintaining API consistency, handling continuous learning and uncertainty in ML models.	Demands developing deployment strategies for responsible AI, maintaining high automation levels in deployment environments. However, best practices include using CI/CD pipelines for continuous deployment, leveraging cloud platforms for scalability, implementing APIs for integration, and ensuring rollback mechanisms for conventional software to enhance deployment reliability.
Maintenance	
Continuous monitoring to detect performance degradation in ML components. There are challenges like quality control of the large datasets[106, 151], automating maintenance process triggered by runtime errors. The ML landscape offers powerful tools for predictions, but often leads to significant ongoing maintenance costs[118].	Implementing techniques like open-set recognition for timely maintenance and support measures are needed[146]. In general, continuous monitoring of performance degradation of the ML component through output monitoring, data drift detection, and performance metrics tracking; using automated retraining schedules[118], version control for model updates, and logging tools to track metrics and user interactions are among the best maintenance practices.
Development methodologies	
Agile frameworks are prevalent in ML projects, fostering flexibility and iterative development [67, 112]. Adapting existing methodologies to capture unique characteristics of ML-enabled software is challenging due to new system requirements and imperfection, uncertainty, and lack of vision among the development team[55].	There is a need for exploring new methodologies specific to the integration of ML and AI ethics within agile. However, best practices include adopting automated frameworks[19] and DevOps[111] for flexibility and iterative development, and facilitating regular feedback loops thereby integrate AI ethics into user stories. Additionally, apply methods like RUDE to achieve reliable and maintainable ML-enabled software[98].