Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process

Nadia Nahar
nadian@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Shurui Zhou
University of Toronto
Toronto, Ontario, Canada

Grace Lewis
Carnegie Mellon Software Engineering Institute
Pittsburgh, PA, USA

Christian Kästner
Carnegie Mellon University
Pittsburgh, PA, USA

ABSTRACT

The introduction of machine learning (ML) components in software projects has created the need for software engineers to collaborate with data scientists and other specialists. While collaboration can always be challenging, ML introduces additional challenges with its exploratory model development process, additional skills and knowledge needed, difficulties testing ML systems, need for continuous evolution and monitoring, and non-traditional quality requirements such as fairness and explainability. Through interviews with 45 practitioners from 28 organizations, we identified key collaboration challenges that teams face when building and deploying ML systems into production. We report on common collaboration points in the development of production ML systems for requirements, data, and integration, as well as corresponding team patterns and challenges. We find that most of these challenges center around communication, documentation, engineering, and process, and collect recommendations to address these challenges.

ACM Reference Format:
Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. 2022. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510209

1 INTRODUCTION

Machine learning (ML) is receiving massive attention and funding in research and practice; it is achieving incredible advances, surpassing human-level cognition in many applications, but it is widely acknowledged that moving from a prototyped machine-learned model to a production system is very challenging. For example, Venturebeat reported in 2019 that 87 percent of ML projects fail [108] and Gartner claimed in 2020 that 53 percent do not make it from prototype to production [70]. While traditional software projects are already complex, failure prone, and require a broad range of expertise, the introduction of machine learning raises further challenges, requires additional expertise, and introduces additional collaboration points.

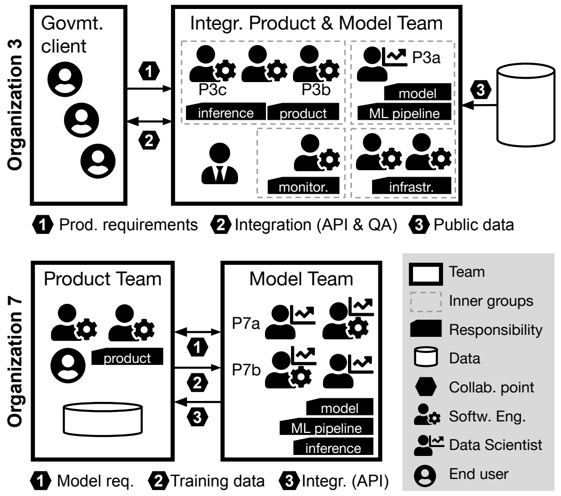


Figure 1: Structure of two interviewed organizations

Technical aspects such as testing ML components [10, 20], misuse of ML libraries [43, 45], engineering challenges for developing ML components [3, 5, 18, 27, 40, 44, 60, 90], and automating learning and deployment processes for ML components [4, 13, 29, 34, 51], have received significant attention in research recently. However, human factors of collaboration during the development of software products supported by ML components, *ML-enabled systems* for short, have received less attention, including the need to separate and coordinate data science and software engineering work, to negotiate and document interfaces and responsibilities, and to plan the system's operation and evolution. Yet, those human collaboration challenges appear to be major hurdles in developing ML-enabled systems. In addition, past work has mostly been model-centric, focused on challenges of learning, testing, or serving models, but rarely focuses on the entire system, i.e., the product with many non-ML parts into which the model is embedded as a component, which requires coordinating and integrating work from multiple experts or teams.

To better understand collaboration challenges and avenues toward better practices, we conducted interviews with 45 participants

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510209

contributing to the development of ML-enabled systems for production use (i.e., not pure data analytics/early prototypes). Our research question is: *What are the collaboration points and corresponding challenges between data scientists and software engineers?* Participants come from 28 organizations, from small startups to large big tech companies, and have diverse roles in these projects, including data scientists, software engineers, and managers. During our interviews, we explored organizational structures (e.g., see Figure 1), interactions of project members with different technical backgrounds, and where conflicts arise between teams.

While some organizations have adopted better collaboration practices than others, many struggle setting up structures, processes, and tooling for effective collaboration among team members with different backgrounds when developing ML-enabled systems. To the best of our knowledge, and confirmed by the practitioners we interviewed, there is little systematic or shared understanding of common collaboration challenges and best practices for developing ML-enabled systems and coordinating developers with very different backgrounds (e.g., data science vs. software engineering). We find that smaller and new-to-ML organizations struggle more, but have limited advice to draw from for improvement.

Three *collaboration points* surfaced as particularly challenging: (1) Identifying and decomposing requirements, (2) negotiating training data quality and quantity, and (3) integrating data science and software engineering work. We found that organizational structure, team composition, power dynamics, and responsibilities differ substantially, but also found common *organizational patterns* at specific collaboration points and challenges associated with them.

Overall, our observations suggest four *themes* that would benefit from more attention when building ML-enabled systems: Invest in supporting *interdisciplinary teams* to work together (including education and avoiding silos), Pay more attention to collaboration points and clearly *document* responsibilities and interfaces, Consider *engineering work* as a key contribution to the project, and Invest more into *process and planning*.

In summary, we make the following contributions: (1) We identify three core collaboration points and associated collaboration challenges based on interviews with 45 practitioners, triangulated with a literature review, (2) We highlight the different ways in which teams organize, but also identify organizational patterns that associate with certain collaboration challenges, and (3) We identify recommendations to improve collaboration practices.

2 STATE OF THE ART

Researchers and practitioners have discussed whether and how machine learning changes software engineering with the introduction of learned models as components in software systems [e.g., 1, 5, 42, 69, 81, 83, 90, 103, 111]. To lay the foundation for our interview study and inform the questions we ask, we first provide an overview of the related work and existing theories on collaboration in traditional software engineering and discuss how machine learning may change this.

Collaboration in Software Engineering. Most software projects exceed the capacity of a single developer, requiring multiple developers and teams to collaborate ("work together") and coordinate ("align goals"). Collaboration happens *across* teams, often in a more formal and structured form, and *within* teams, where familiarity with other team members and frequent co-location fosters informal communication [63]. At a technical level, to allow multiple developers to work together, *abstraction* and a *divide and conquer* strategy are essential. Dividing software into *components* (modules, functions, subsystems) and hiding internals behind *interfaces* is a key principle of modular software development that allows teams to divide work, and work mostly independently until the final system is integrated [62, 72].

Teams within an organization tend to align with the technical structure of the system, with individuals or teams assigned to components [30], hence the technical structure (interfaces and dependencies between components) influences the points where teams collaborate and coordinate. Coordination challenges are especially observed when teams cannot easily and informally communicate, often studied in the context of distributed teams of global corporations [38, 68] and open-source ecosystems [16, 95].

More broadly, *interdisciplinary* collaboration often poses challenges. It has been shown that when team members differ in their academic and professional backgrounds and possess different expectations on the same system, communication, cultural, and methodical challenges often emerge when working together [21, 73]. Key insights are that successful interdisciplinary collaboration depends on professional role, structural characteristics, personal characteristics, and a history of collaboration; specifically, structural factors such as unclear mission, insufficient time, excessive workload, and lack of administrative support are barriers to collaboration [24].

The component *interface* plays a key role in collaboration as a *negotiation and collaboration point.* It is where teams (re-)negotiate how to divide work and assign responsibilities [19]. Team members often seek information that may not be captured in interface descriptions, as interfaces are rarely fully specified [32]. In an idealized development process, interfaces are defined early based on what is assumed to remain stable [72], because changes to interfaces later are expensive and require the involvement of multiple teams. In addition, interfaces reflect *key architectural decisions* for the system, aimed to achieve desired overall qualities [11].

In practice though, the idealized divide-and-conquer approach following top-down planning does not always work without friction. Not all changes can be anticipated, leading to later modifications and renegotiation of interfaces [16, 31]. It may not be possible to identify how to decompose work and design stable interfaces until substantial experimentation has been performed [12]. To manage, negotiate, and communicate changes of interfaces, developers have adopted a wide range of strategies for communication [16, 33, 97], often relying on informal broadcast mechanisms to share planned or performed changes with other teams.

Software lifecycle models [22] also address this tension of when and how to design stable interfaces: Traditional top-down models (e.g., waterfall) plan software design after careful requirements analysis; the *spiral model* pursues a risk-first approach in which developers iterate to prototype risky parts, which then informs future system design iterations; *agile* approaches de-emphasize upfront architectural design for fast iteration on incremental prototypes. The software architecture community has also grappled with the question of how much upfront architectural design is feasible, practical, or desirable [11, 107], showing a tension between the desire for upfront planning on one side and technical risks and unstable requirements on the other. *In this context, our research explores how introducing machine learning into software projects challenges collaboration.*

Software Engineering with ML Components. In a ML-enabled system, machine learning contributes one or multiple components to a larger system with traditional non-ML components. We refer to the whole system that an end user would use as the *product*. In some systems, the learned *model* may be a relatively small and isolated addition to a large traditional software system (e.g., audit prediction in tax software); in others it may provide the system's essential core with only minimal non-ML code around it (e.g., a sales prediction system sending daily predictions by email). In addition to models, an ML-enabled system typically also has components for training and monitoring the model(s) [42, 51]. Much attention in practice recently focuses on building robust ML *pipelines* for training and deploying models in a scalable fashion, often under names such as *"AI engineering," "SysML,"* and *"MLOps"* [51, 59, 67, 90]. In this work, we focus more broadly on the development of the entire ML-enabled system, including both ML and non-ML components.

Compared to traditional software systems, ML-enabled systems require additional expertise in *data science* to build the models and may place additional emphasis on expertise such as data management, safety, and ethics [5, 49]. In this paper, we primarily focus on the roles of *software engineers* and *data scientists*, who typically have different skills and educational backgrounds [48, 49, 84, 111]: Data science education tends to focus more on statistics, ML algorithms, and practical training of models from data (typically given a fixed dataset, not deploying the model, not building a system), whereas software engineering education focuses on engineering tradeoffs with competing qualities, limited information, limited budget, and the construction and deployment of systems. Research shows that software engineers who engage in data science without further education are often naive when building models [111] and that data scientists prefer to focus narrowly on modeling tasks [84] but are frequently faced with engineering work [106]. While there is plenty of work on supporting collaboration among software engineers [26, 33, 85, 115] and more recently on supporting collaboration among data scientists [105, 114], we are not aware of work exploring collaboration challenges between these roles, which we do in this work.

The software engineering community has recently started to explore *software engineering for ML-enabled systems* as a research field, with many contributions on bringing software-engineering techniques to ML tasks, such as testing models and ML algorithms [10, 20, 28, 110], deploying models [4, 13, 29, 34, 51], robustness and fairness of models [81, 94, 101], life cycles for ML models [1, 5, 34, 61, 74], and engineering challenges or best practices for developing ML components [3, 5, 18, 27, 40, 44, 60, 90]. A smaller body of work focuses on the ML-enabled system beyond the model, such as exploring system-level quality attributes [73, 93], requirements engineering [103], architectural design [113], safety mechanisms [17, 83], and user interaction design [7, 25, 112]. *In this paper, we adopt this system-wide scope and explore how data scientists and software engineers work together to build the system with ML and non-ML components.*

3 RESEARCH DESIGN

Because there is limited research on collaboration in building ML-enabled systems, we adopt a qualitative research strategy to explore *collaboration points* and corresponding *challenges*, primarily with stakeholder interviews. We proceeded in four steps: (1) We prepared interviews based on an initial literature review, (2) we conducted interviews, (3) we triangulated results with literature findings, and (4) we validated our findings with the interview participants. We base our research design on *Straussian Grounded Theory* [98, 99], which derives research questions from literature, analyzes interviews with open and axial coding, and consults literature throughout the process. In particular, we conduct interviews and literature analysis in parallel, with immediate and continuous data analysis, performing constant comparisons, and refining our codebook and interview questions throughout the study.

Step 1: Scoping and interview guide. To scope our research and prepare for interviews, we looked for collaboration problems mentioned in existing literature on software engineering for ML-enabled systems (Sec. 2). In this phase, we selected 15 papers opportunistically through keyword search and our own knowledge of the field. We marked all sections in those papers that potentially relate to collaboration challenges between team members with different skills or educational backgrounds, following a standard open coding process [99]. Even though most papers did not talk about problems in terms of collaboration, we marked discussions that may plausibly relate to collaboration, such as data quality issues between teams. We then analyzed and condensed these codes into nine initial collaboration areas and developed an initial codebook and interview guide (provided in Appendix of arXiv version [66]).

Step 2: Interviews. We conducted semi-structured interviews with 45 participants from 28 organizations, each 30 to 60 minutes long. All participants are involved in professional software projects using machine learning that are either already or planned to be deployed in production. In Table 1, we show the demographics of the interview participants and their organizations. Details can be found in the Appendix of our arXiv version [66].

We tried to sample participants purposefully (maximum variation sampling [36]) to cover participants in different roles, types of companies, and countries. We intentionally recruited most participants from organizations outside of big tech companies, as they represent the vast majority of projects that have recently adopted machine learning and often face substantially different challenges [40]. Where possible, we tried to separately interview multiple participants in different roles within the same organization to get different perspectives. We identified potential participants through personal networks, ML-related networking events, LinkedIn, and recommendations from previous interviewees and local tech leaders. We adapted our recruitment strategy throughout the research based on our findings, at later stages focusing primarily on specific roles and organizations to fill gaps in our understanding, until reaching saturation. For confidentiality, we refer to organizations by number and to participants by P*Xy* where *X* refers to the organization number and *y* distinguishes participants from the same organization.

Table 1: Participant and Company Demographics

Type	Break-down
Participant Role (45)	ML-focused (23), SE-focused (9), Management (5), Operations (2), Domain Expert (2), Other (4)
Participant Seniority (45)	5 years of experience or more (28), 2-5 years (9), under 2 years (8)
Company Type (28)	Big tech (6), Non IT (4), Mid-size tech (11), Startup (5), Consulting (2)
Company Location (28)	North America (11), South America (1), Europe (5), Asia (10), Africa (1)

We transcribed and analyzed all interviews. Then, to map challenges to collaboration points, we created visualizations of organizational structure and responsibilities in each organization (we show two examples in Figure 1) and mapped collaboration problems mentioned in the interviews to collaboration points within these visualizations. We used these visualizations to further organize our data; in particular, we explored whether collaboration problems associate with certain types of organizational structures.

Step 3: Triangulation with literature. As we gained insights from interviews, we returned to the literature to identify related discussions and possible solutions (even if not originally framed in terms of collaboration) to triangulate our interview results. Relevant literature spans multiple research communities and publication venues, including machine learning, human-computer interaction, software engineering, systems, and various application domains (e.g., healthcare, finance), and does not always include obvious keywords; simply searching for machine-learning research yields a far too wide net. Hence, we decided against a systematic literature review and pursued a best effort approach that relied on keyword search for topics surfaced in the interviews, as well as backward and forward snowballing. Out of over 300 papers read, we identified 61 as possibly relevant and coded them with the same evolving codebook. The complete list can be found in our arXiv version [66].

Step 4: Validity check with interviewees. For checking fit and applicability as defined by Corbin and Strauss [99] and validating our findings, we went back to the interviewees after creating a full draft of this paper. We presented the interviewees both a summary and the full draft, including the supplementary material, along with questions prompting them to look for correctness and areas of agreement or disagreement (i.e., fit), and any insights gained from reading about experiences of the other companies, roles, or findings as a whole (i.e., applicability). Ten interviewees responded with comments and all indicated general agreement, some explicitly reaffirmed some findings. We incorporated two minor suggested changes about details of two organizations.

Threats to validity and credibility. Our work exhibits the typical threats common and expected for this kind of qualitative research. Generalizations beyond the sampled participant distribution should be made with care; for example, we interviewed few managers, no dedicated data experts, and no clients. In several organizations, we were only able to interview a single person, giving us a one-sided perspective. Observations may be different in organizations in specific domains or geographic regions not well represented in our data. Self-selection of participants may influence results; for example developers in government-related projects more frequently declined interview requests. As described earlier, we followed standard practices for coding and memoing, but, as usual in qualitative research, we cannot entirely exclude biases introduced by us researchers.

4 DIVERSITY OF ORG. STRUCTURES

Throughout our interviews, we found that the number and type of teams that participate in ML-enabled system development differs widely, as do their composition and responsibilities, power dynamics, and the formality of their collaborations, in line with findings by Aho et al. [1]. To illustrate these differences, we provide simplified descriptions of teams found in two organizations in Figure 1. We show teams and their members, as well as the artifacts for which they are *responsible*, such as, who develops the *model*, who builds a repeatable *pipeline*, who operates the model (*inference*), who is responsible for or owns the *data*, and who is responsible for the final *product*. A team often has multiple responsibilities and interfaces with other teams at multiple collaboration points. Where unambiguous, we refer to teams by their primary responsibility as *product team* or *model team*.

Organization 3 (Figure 1, top) develops an ML-enabled system for a government client. The product (health domain), including an ML model and multiple non-ML components, is developed by a single 8-person team. The team focuses on training a model first, before building a product around it. Software engineering and data science tasks are distributed within the team, where members cluster into groups with different responsibilities and roughly equal negotiation power. A single data scientist is part of this team, though they feel somewhat isolated. Data is sourced from public sources. The relationship between the client and development team is somewhat distant and formal. The product is delivered as a service, but the team only receives feedback when things go wrong.

Organization 7 (Figure 1, bottom) develops a product for in-house use (quality control for a production process). A small team is developing and using the product, but model development is delegated to an external team (different company) composed of four data scientists, of which two have some software engineering background. The product team interacts with the model team to define and revise model requirements based on product requirements. The product team provides confidential proprietary data for training. The model team deploys the model and provides a ready-to-use inference API to the product team. The relationship between the teams crosses company boundaries and is rather distant and formal. The product team clearly has the power in negotiations between the teams.

These two organizations differed along many dimensions, and we found no clear global patterns when looking across organizations. Nonetheless patterns did emerge when focusing on three specific collaboration aspects, as we will discuss in the next sections.

5 COLLABORATION POINT: REQUIREMENTS AND PLANNING

In an idealized top-down process, one would first solicit *product requirements* and then plan and design the product by dividing work into components (ML and non-ML), deriving each *component’s requirements/specifications* from the product requirements. In this process, collaboration is needed for: (1) product team needs to negotiate *product* requirements with clients and other stakeholders; (2) product team needs to plan and design product decomposition, negotiating with component teams the requirements for individual components; and (3) product project manager needs to plan and manage the work across teams in terms of budgeting, effort estimation, milestones, and work assignments.

5.1 Common Development Trajectories

Few organizations, if any, follow an idealized top-down process, and it may not even be desirable, as we will discuss later. While we did not find any global patterns for organizational structures (Sec. 4), there are indeed distinct patterns relating to how organizations elicit requirements and decompose their systems. Most importantly, we see differences in terms of the *order* in which teams identify product and model requirements:

Model-first trajectory: 13 of the 28 organizations (3, 10, 14–17, 19, 20, 22, 23, 25–27) focus on building the model first, and build a product around the model later. In these organizations, product requirements are usually shaped by model capabilities after the (initial) model has been created, rather than being defined upfront. In organizations with separate model and product teams, the model team typically starts the project and the product team joins later with low negotiating power to build a product around the model.

Product-first trajectory: In 13 organizations (1, 4, 5, 7–9, 11–13, 18, 21, 24, 28), models are built later to support an existing product. In these cases, a product often already exists and product requirements are collected for how to extend the product with new ML-supported functionality. Here, the model requirements are derived from the product requirements and often include constraints on model qualities, such as latency, memory and explainability.

Parallel trajectory: Two organizations (2, 6) follow no clear temporal order; model and product teams work in parallel.

5.2 Product and Model Requirements

We found a constant tension between product and model requirements in our interviews. Functional and nonfunctional product requirements set expectations for the entire product. Model requirements set goals and constraints for the model team, such as expected accuracy and latency, target domain, and available data.

Product requirements require input from the model team (👥, 📅). A common theme in the interviews is that it is difficult to elicit product requirements without a good understanding of ML capabilities, which almost always requires involving the model team and performing some initial modeling when eliciting product requirements. Regardless of whether product requirements or model requirements are elicited first, data scientists often mentioned being faced with *unrealistic expectations* about model capabilities.

Participants that interact with clients to negotiate product requirements (which may involve members of the model team) indicate that they need to educate clients about capabilities of ML techniques to *set correct expectations* (P3a, P6a, P6b, P7b, P9a, P10a, P15c, P19b, P22b, P24a). This need to educate customers about ML capabilities has also been raised in the literature [1, 17, 44, 49, 100, 103, 106].

For many organizations, especially in *product-first trajectories*, the model team indicates similar challenges when interacting with the product team. If the product team does not involve the model team in negotiating product requirements, the product team may not identify what data is needed for building the model, and may commit to unrealistic requirements. For example, P26a shared *“For this project, [the project manager] wanted to claim that we have no false positives and I was like, that’s not gonna work.”* Members of the model team often report lack of ML literacy in members of the product team and project managers (P1b, P4a, P7a, P12a, P26a, P27a) and a lack of involvement (e.g., P7b: “*The [product team] decided what type of data would make sense. I had no say on that.”*). Usually the product team cannot identify product requirements alone, instead product and model teams need to interact to explore what is achievable.

In organizations with a *model-first trajectory*, members of the model team sometimes engage directly with clients (and also report having to educate them about ML capabilities). However, when requirements elicitation is left to the model team, members tend to focus on requirements relevant for the model, but neglect requirements for the product, such as expectations for usability, e.g., P3c’s customers “*were kind of happy with the results, but weren’t happy with the overall look and feel or how the system worked.”* Several research papers similarly identified how the goals of data scientists diverge from product goals if product requirements are not obvious at modeling time, leading to inefficient development, worse products, or constant renegotiation of requirements, especially [67, 73, 112].

Model development with unclear model requirements is common (📄). Participants from model teams frequently explain how they are expected to work independently, but are given sparse model requirements. They try to infer intentions behind them, but are constrained by having limited understanding of the product that the model will eventually support (P3a, P3b, P16b, P17b, P19a). Model teams often start with vague goals and model requirements evolve over time as product teams or clients refine their expectations in response to provided models (P3b, P7a, P9a, P5b, P19b, P21a). Especially in organizations following the *model-first trajectory*, model teams may receive some data and a goal to predict something with high accuracy, but no further context, e.g., P3a shared *“there isn’t always an actual spec of exactly what data they have, what data they think they’re going to have and what they want the model to do.”* Several papers similarly report projects starting with vague model goals [50, 77, 83, 111].

Even in organizations following a *product-first trajectory*, product requirements are often not translated into clear model requirements. For example, participant P17b reports how the model team was not clear about the model’s intended target domain, thus could not decide what data was considered in scope. As a consequence, model teams usually cannot focus just on their component, but have to understand the entire product to identify model requirements in the context of the product (P3a, P10a, P13a, P17a, P17b, P19b, P20b, P23a), requiring interactions with the product team or even bypassing the product team to talk directly to clients. The difficulty of providing clear requirements for an ML model has also been raised in the literature [49, 55, 80, 92, 104, 111], partially arguing that uncertainty makes it difficult to specify model requirements upfront [1, 44, 50, 69, 106]. Ashemore et al. report mapping product requirements to model requirements as an open challenge [10].

Provided model requirements rarely go beyond accuracy and data security (⚙, 📄). Requirements given to model teams primarily relate to some notion of accuracy. Beyond accuracy, requirements for data security and privacy are common, typically imposed by the data owner or by legal requirements (P5a, P7a, P9a, P13a, P14a, P18a, P20a-b, P21a-b, P22a, P23a, P24a, P25a, P26a). Literature also frequently discusses how privacy requirements impact and restrict ML work [15, 41, 43, 55, 56, 78].

We rarely heard of any qualities other than accuracy. Some participants report that ignoring qualities such as latency or scalability has resulted in integration and operation problems (P3c, P11a). In a few cases requirements for inference latency were provided (P1a, P6a, P14a) and in one case hardware resources provided constraints on memory usage (P14a), but no other qualities such as training latency, model size, fairness, or explainability were required that could be important for product integration and deployment.

When prompted, very few of our interviewees report considerations for fairness either at the product or the model level. Only two participants from model teams (P14a, P22a) reported receiving fairness requirements, whereas many others explicitly mentioned that fairness is not a concern for them yet (P4a, P5b, P6b, P11a, P15c, P20a, P21b, P25a, P26a). The lack of fairness and explainability requirements is in stark contrast to the emphasis that these qualities receive in the literature [e.g., 7, 15, 25, 39, 40, 57, 89, 92, 109, 114].

Recommendations. Our observations suggest that involving data scientists early when soliciting product requirements is important (👥) and that pursuing a model-first trajectory entirely without considering product requirements is problematic (📅). Conversely, model requirements are rarely specific enough to allow data scientists to work in isolation without knowing the broader context of the system and interaction with the product team should ideally be planned as part of the process. Requirements form a key collaboration point between product and model teams, which should be emphasized even in more distant collaboration styles (e.g., outsourced model development). The few organizations that use the *parallel trajectory* report fewer problems by involving data scientists in negotiating product requirements to discard unrealistic ones early on (P6b). Vogelsang and Borg also provide similar recommendations to consult data scientists from the beginning to help elicit requirements [103]. While many papers place emphasis on clearly defining ML use cases and scope [49, 93, 100], several others mention how collaboration of technical and non-technical stakeholders such as domain experts helps [73, 89, 104, 106].

ML literacy for customers and product teams appears to be important (👥). P22a and P19a suggested conducting technical ML training sessions to educate clients; similar training is also useful for members of product teams. Several papers argue for similar training for non-technical users of ML products [44, 89, 103].

Most organizations elicit requirements only rather informally and rarely have good documentation, especially when it comes to model requirements. It seems beneficial to adopt more formal requirements documentation for product and model (📄), as several participants reported that it fosters shared understanding at this collaboration point (P11a, P13a, P19b, P22a, P22c, P24a, P25a, P26a). Checklists could help to cover a broader range of model quality requirements, such as training latency, fairness, and explainability. Formalisms such as model cards [64] and FactSheets [8] could be used as a starting point for documenting model requirements.

5.3 Project Planning

ML uncertainty makes effort estimation difficult (👥). Irrespective of trajectory, 19 participants (P3a, P4a, P7a-b, P8a, P14b, P15b-c, P16a, P17a, P18a, P19a-b, P20a, P22a-c, P23a, P25a) mentioned that the uncertainty associated with ML components makes it difficult to estimate the timeline for developing an ML component and by extension the product. Model development is typically seen as a science-like activity, where iterative experimentation and exploration is needed to identify whether and how a problem can be solved, rather than as an engineering activity that follows a somewhat predictable process. This science-like nature makes it difficult for the model team to set expectations or contracts with clients or the product team regarding effort, cost, or accuracy. While data scientists find effort estimation difficult, lack of ML literacy in managers makes it worse (P15b, P16a, P19b, P20a, P22b). Teams report deploying subpar models when running out of time (P3a, P15b, P19a), or postponing or even canceling deployments (P25a). These findings align with literature mentioning difficulties associated with effort estimation for ML tasks [1, 9, 61, 106] and planning projects in a structured manner with diverse methodologies, with diverse trajectories, and without practical guidance [1, 17, 61, 106].

Generally, participants frequently report that synchronization between teams is challenging because of different team pace, different development processes, and tangled responsibilities (P2a, P11a, P12a, P14-b, P15b-c, P19a; see also Sec. 7.2).

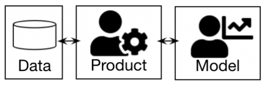
Recommendations. Participants suggested several mitigation strategies: keeping extra buffer times and adding additional timeboxes for R&D in initial phases (P8a, P19a, P22b-c, P23a; 📅), continuously involving clients in every phase so that they can understand the progression of the project and be aware of potential missed deadlines (P6b, P7a, P22a, P23a; 👥). From the interviews, we also observe the benefits of managers who understand both software engineering and machine learning and can align product and model teams toward common goals (P2a, P6a, P8a, P28a; 👥).

6 COLLABORATION POINT: TRAINING DATA

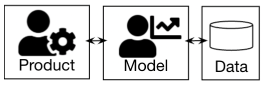
Data is essential for machine learning, but disagreements and frustrations around training data were the most common collaboration challenges mentioned in our interviews. In most organizations, the team that is responsible for building the model is not the team that collects, owns, and understands the data, making data a key collaboration point between teams in ML-enabled systems development.

6.1 Common Organizational Structures

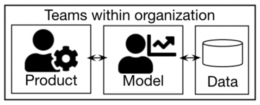
We observed three patterns around data that influence collaboration challenges from the perspective of the model team:



Provided data: The product team has the responsibility of providing data to the model team (org. 6–8, 13, 18, 21, 23). The product team is the initial point of contact for all data-related questions from the model team. The product team may own the data or acquire it from a separate data team (internal or external). Coordination regarding data tends to be distant and formal, and the product team tends to hold more negotiation power.



External data: The product team does not have direct responsibility for providing data, but instead, the model team relies on external data providers. Commonly, the model team (i) uses publicly available resources (e.g., academic datasets, org. 2–4, 6, 19) or (ii) hires a third party for collecting or labeling data (org. 9, 15–17, 22, 23). In the former case, the model team has little to no negotiation power over data; in the latter, it can set expectations.



In-house data: Product, model, and data teams are all part of the same organization and the model team relies on internal data from that organization (org. 1, 5, 9–12, 14, 20, 24–28). In these cases, both product and model teams often find it challenging to negotiate access to internal data due to differing priorities, internal politics, permissions, and security constraints.

6.2 Negotiating Data Quality and Quantity

Disagreements and frustrations around training data were the most common collaboration challenges in our interviews. In almost every project, data scientists were unsatisfied with the quality and quantity of data they received at this collaboration point, in line with a recent survey showing data availability and management to be the top-ranked challenge in building ML-enabled systems [5].

Provided and public data is often inadequate (📄, 👥). In organizations where data is provided by the product team, the model team commonly states that it is difficult to get sufficient data (P7a, P8a, P13a, P22a, P22c). The data that they receive is often of low quality, requiring significant investment in data cleaning. Similar to the requirements challenges discussed earlier, they often state that the product team has little knowledge or intuition for the amount and quality of data needed. For example, participant P13a stated that they were given a spreadsheet with only 50 rows to build a model and P7a reported having to spend a lot of time convincing the product team of the importance of data quality. This aligns with past observations that software engineers often have little appreciation for data quality concerns [49, 54, 65, 77, 84] and that training data is often insufficient and incomplete [6, 43, 56, 77, 83, 93, 106].

When the model team uses public data sources, its members also have little influence over data quality and quantity and report significant effort for cleaning low quality and noisy data (P2a, P3a, P4a, P3c, P6b, P19b, P23a). Papers have similarly questioned the representativeness and trustworthiness of public training data [34, 103, 109] as *"nobody gets paid to maintain such data"* [104].

Training-serving skew is a common challenge when training data is provided to the model team: models show promising results, but do not generalize to production data because it differs from provided training data (P4a, P8a, P13a, P15a, P15c, P21a, P22c, P23a) [9, 23, 55, 56, 77–79, 84, 100, 109, 116]. Our interviews show that this skew often originates from inadequate training data combined with unclear information about production data, and therefore no chance to evaluate whether the training data is representative of production data.

Data understanding and access to domain experts is a bottleneck (📄, 📅). Existing data documentation (e.g, data item definitions, semantics, schema) is almost never sufficient for model teams to understand the data (also mentioned in a prior study [46]). In the absence of clear documentation, team members often collect information and keep track of unwritten details in their heads (P5a), known as institutional or tribal knowledge [5, 40]. Data understanding and debugging often involve members from different teams and thus cause challenges at this collaboration point.

Model teams receiving data from the product team report struggling with data understanding and having a difficult time getting help from the product team (or the data team that the product team works with) (P8a, P7b, P13a). As the model team does not have direct communication with the data team, data understanding issues often cannot be resolved effectively. For example, P13a reports *"Ideally, for us it would be so good to spend maybe a week or two with one person continuously trying to understand the data. It's one of the biggest problems actually, because even if you have the person, if you're not in contact all the time, then you misinterpreted some things and you build on it."* The low negotiation power of the model team in these organizations hinders access to domain experts.

Model teams using public data similarly struggle with data understanding and getting help (P3a, P4a, P19a), relying on sparse data documentation or trying to reach any experts on the data.

For in-house projects, in several organizations the model team relies on data in shared databases (org. 5, 11, 26, 27, 28), collected by instrumenting a production system, but shared by multiple teams. Several teams shared problems with evolving and often poorly documented data sources, as participant P5a illustrates *"[data rows] can have 4,000 features, 10,000 features. And no one really cares. They just dump features there. [...] I just cannot track 10,000 features."* Model teams face challenges in understanding data and identifying a team that can help (P5a, P25a, P20b, P27a), a problem also reported in a prior study about data scientists at Microsoft [49].

Challenges in understanding data and needing domain experts are also frequently mentioned in the literature [13, 40, 41, 46, 49, 65, 77, 84], as is the danger of building models with insufficient understanding of the data [34, 103]. Although we are not aware of literature discussing the challenges of accessing domain experts, papers have shown that even when data scientists have access, effective knowledge transfer is challenging [71, 91].

Ambiguity when hiring a data team (📄). When the model team hires an external data team for collecting or labelling data (org. 9, 15, 16, 17, 22, 23), the model team has much more negotiation power over setting data quality and quantity expectations (though Kim et al. report that model teams may have difficulty getting buy-in from the product team for hiring a data team in the first place [49]). Our interviews did not surface the same frustrations as with provided data and public data, but instead participants from these organizations reported *communication vagueness* and *hidden assumptions* as key challenges at this collaboration point (P9a, P15a, P15c, P16a, P17b, P22a, P22c, P23a). For example, P9a related how different labelling companies given the same specification widely disagreed on labels, when the specification was not clear enough.

We found that expectations between model and data teams are often communicated verbally without clear documentation. As a result, the data team often does not have sufficient context to understand what data is needed. For example, participant P17b states *"Data collectors can't understand the data requirements all the time. Because, when a questionnaire [for data collection] is designed, the overview of the project is not always described to them. Even if we describe it, they can't always catch it."* Reports about low quality data from hired data teams have been also discussed in the literature [10, 43, 55, 84, 103, 106].

Need to handle evolving data (⚙, 👥). In most projects, models need to be regularly retrained with more data or adapted to changes in the environment (e.g., data drift) [42, 55, 84], which is a challenge for many model teams (P3a, P3c, P5a, P7a-b, P11a, P15c, P18a, P19b, P22a). When product teams provide the data, they often have a static view and provide only a single snapshot of data rather than preparing for updates, and model teams with their limited negotiation power have a difficult time fostering a more dynamic mindset (P7a-b, P15c, P18a, P22a), as expressed by participant P15c: *"People don't understand that for a machine learning project, data has to be provided constantly."* It can be challenging for a model team to convince the product team to invest in continuous model maintenance and evolution (P7a, P15c) [46].

Conversely, if data is provided continuously (most commonly with public data sources, in-house sources, and own data teams), model teams struggle with ensuring consistency over time. Data sources can suddenly change without announcement (e.g., changes to schema, distributions, semantics), surprising model teams that make but do not check assumptions about the data (P3a, P3c, P19b). For example, participants P5a and P11a report similar challenges with in-house data, where their low negotiation power does not allow them to set quality expectations, but they face undesired and unannounced changes in data sources made by other teams. Most organizations do not have a monitoring infrastructure to detect changes in data quality or quantity, as we will discuss in Sec. 7.3.

In-house priorities and security concerns often obstruct data access (📅). In in-house projects, we frequently heard about the product or model team struggling to work with another team within the same organization that owns the data. Often, these in-house projects are local initiatives (e.g., logistics optimization) with more or less buy-in from management and without buy-in from other teams that have their own priorities; sometimes other teams explicitly question the business value of the product. The interviewed model teams usually have little negotiation power to request data (especially if it involves collecting additional data) and almost never get an agreement to continuously receive data in a certain format, quality, or quantity (P5a, P10a, P11a, P20a-b, P27a) (also observed in studies at Microsoft, ING and other organizations [34, 49, 65]). For example, P10a shared *"we wanted to ask the data warehouse team to [provide data], and it was really hard to get resources. They wouldn't do that because it was hard to measure the impact [our in-house project] had on the bottom line of the business."* Model teams in these settings tend to work with whatever data they can get eventually.

Security and privacy concerns can also limit access to data (P7a, P7b, P21a-b, P22a, P24a) [46, 55, 56, 65, 77], especially when data is owned by a team in a different organization, causing frustration, lengthy negotiations, and sometimes expensive data-handling restrictions (e.g., no use of cloud resources) for model teams.

Recommendations. Data quality and quantity is important to model teams, yet they often find themselves in a position of low negotiation power, leading to frustration and collaboration inefficiencies. Model teams that have the freedom to set expectations and hire their own data teams are noticeably more satisfied. When planning the entire product, it seems important to pay special attention to this collaboration point, and budget for data collection, access to domain experts, or even a dedicated data team (📅). Explicitly planning to provide substantial access to domain experts early in the project was suggested as important (P25a).

We found it surprising that despite the importance of this collaboration point there is little written agreement on expectations and often limited documentation (📄), even when hiring a dedicated data team—in stark contrast to more established contracts for traditional software components. Not all organizations allow the more agile, constant close collaboration between model and data teams that some suggest [77, 79]. With a more formal or distant relationship (e.g., across organizations, teams without buy-in), it seems beneficial to adopt a *more formal contract*, specifying data quantity and quality expectations, which are well researched in the database literature [58] and have been repeatedly discussed in the context of ML-enabled systems [43, 46, 49, 56, 91]. This has also been framed as *data requirements* in the software engineering literature [83, 100, 103]. When working with a dedicated data team, participants suggested to invest in making expectations very clear, for example, by providing precise specifications and guidelines (P9a, P6b, P28a), running training sessions for the data collectors and annotators (P17b, P22c), and measuring inter-rater agreement (P6b).

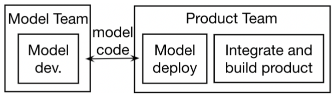
Automated checks are also important as data evolves (⚙). For example, participant P13a mentioned proactively setting up data monitoring to detect problems (e.g., schema violations, distribution shifts) at this collaboration point; a practice suggested also in the literature [53, 56, 77, 79, 84, 89, 100] and supported by recent tooling [e.g., 47, 79, 86]. The risks regarding possible unnoticed changes to data make it important to consider *data validation* and *monitoring infrastructure* as a key feature of the product early on (⚙, 📅), as also emphasized by several participants (P5a, P25a, P26a, P28a).

7 COLLABORATION POINT: PRODUCT-MODEL INTEGRATION

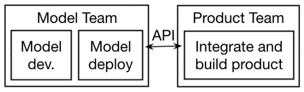
As discussed earlier, to build an ML-enabled system both ML components and traditional non-ML components need to be integrated and deployed, requiring data scientists and software engineers to work together, typically across multiple teams. We found many conflicts at this collaboration point, stemming from unclear processes and responsibilities, as well as differing practices and expectations.

7.1 Common Organizational Structures

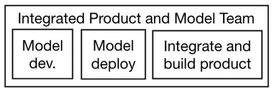
We saw large differences among organizations in how engineering responsibilities were assigned, most visible in how responsibility for *model deployment and operation* is assigned, which typically involves significant engineering effort for building reproducible pipelines, API design, or cloud deployment, often with MLOps technologies. We found the following patterns:



Shared model code: In some organizations (2, 6, 23, 25), the model team is responsible only for model development and delivers training code (e.g., in a notebook) or model files to the product team; the product team takes responsibility for deployment and operation of the model, possibly rewriting the training code as a pipeline. Here, the model team has little or no engineering responsibilities.



Model as API: In most organizations (18 out of 28), the model team is responsible for developing and deploying the model. Hence, the model team requires substantial engineering skills in addition to data science expertise. Here, some model teams are mostly composed of data scientists with little engineering capabilities (org. 7, 13, 17, 22, 26), some consist mostly of software engineers who have picked up some data science knowledge (org. 4, 15, 16, 18, 19, 21, 24), and others have mixed team members (org. 1, 9, 11, 12, 14, 28). These model teams typically provide an API to the product team, or release individual model predictions (e.g., shared files, email; org. 17, 19, 22) or install models directly on servers (org. 4, 9, 12).


Integrated Product and Model Team
Model dev.
Model deploy
Integrate and build product

All-in-one: If only few people work on model and product, sometimes a single team (or even a single person) shares all responsibilities (org. 3, 5, 10, 20, 27). It can be a small team with only data scientists (org. 10, 20, 27) or mixed teams with data scientists and software engineers (org. 3, 5).

We also observed two outliers: One startup (org. 8) had a distinct model deployment team, allowing the model team to focus on data science without much engineering responsibility. In one large organization (org. 28), an engineering-focused model team (model as API) was supported by a dedicated *research team* focused on data-science research with fewer engineering responsibilities.

7.2 Responsibility and Culture Clashes

Interdisciplinary collaboration is challenging (cf. Sec. 2). We observed many conflicts between data science and software engineering culture, made worse by unclear responsibilities and boundaries.

Team responsibilities often do not match capabilities and preferences (⚙). When the model team has responsibilities requiring substantial engineering work, we observed some dissatisfaction when its members were assigned undesired responsibilities. Data scientists preferred engineering support rather than needing to do everything themselves (P7a-b, 13a), but can find it hard to convince management to hire engineers (P10a, P20a, P20b). For example P10a describes *"I was struggling to change the mindset of the team lead, convincing him to hire an engineer...I just didn't want this to be my main responsibility."* Especially in small teams, data scientists report struggling with the complexity of the typical ML infrastructure (P7b, P9a, P14a, P26a, P28a).

In contrast, when deployment is the responsibility of software engineers in the product team or of dedicated engineers in *all-in-one* teams, some of those engineers report problems integrating the models due to insufficient knowledge on model context or domain, and the model code not being packaged well for deployment (P20b, P23a, P27a). In several organizations, we heard about software engineers performing ML tasks without having enough ML understanding (P5a, P15b-c, P16b, 18b, 19b, 20b). Mirroring observations from past research [111], P5a reports *"there are people who are ML engineers at [company] , but they don't really understand ML. They were actually software engineers... they don't understand [overfitting, underfitting, ...]. They just copy-paste code."*

Siloing data scientists fosters integration problems (👥, 📅). We observed data scientists often working in isolation—known as *siloing*—in all types of organizational structures, even within single small teams (see Sec. 4) and within engineering-focused teams. In such settings, data scientists often work in isolation with weak requirements (cf. Sec. 5.2) without understanding the larger context, seriously engaging with others only during integration (P3a, P3c, P6a, P7b, P11a, P13a, P15b, P25a) [41], where problems may surface. For example, participant P11a reported a problem where product and model teams had different assumptions about the expected inputs and the issue could only be identified after a lot of back and forth between teams at a late stage in the project.

Technical jargon challenges communication (👥). Participants frequently described communication issues arising from differing terminology used by members from different backgrounds (P1a-b, P2a, P3a, P5b, P8a, P12a, P14a-b, P16a, P17a-b, P18a-b, P20a, P22b, P23a), leading to ambiguity, misunderstandings, and inconsistent assumptions (on top of communication challenges with domain experts) [1, 46, 76, 104]. P1b reports, *"There are a lot of conversations in which disambiguation becomes necessary. We often use different kinds of words that might be ambiguous."* For example, data scientists may refer to prediction accuracy as *performance*, a term many software engineers associate with response time. These challenges can be observed more frequently between teams, but they even occur within a team with members from different backgrounds (P3a-c, P20a).

Code quality, documentation, and versioning expectations differ widely and cause conflicts (👥, ⚙). Many participants reported conflicts around development practices between data scientists and software engineers during integration and deployment. Participants report poor practices that may also be observed in traditional software projects; but particularly software engineers expressed frustration in interviews that data scientists do not follow the same development practices or have the same quality standards when it comes to writing code. Reported problems relate to poor code quality (P1b, P2a, P3b, P5a, P6a-b, P10a, P11a, P14a, P15b-c, P17a, P18a, P19a, P20a-b, P26a) [9, 27, 34, 37, 75, 87, 106], insufficient documentation (P5a-b, P6a-b, P10a, P15c, P26a) [8, 46, 64, 114], and not extending version control to data and models (P3c, P7a, P10a, P14a, P20b). In two shared-model-code organizations, participants report having to rewrite code from the data scientists (P2a, P6a-b). Missing documentation for ML code and models is considered the cause for different assumptions that lead to incompatibility between ML and non-ML components (P10a) and for losing knowledge and even the model when faced with turnover (P6a-b). Recent papers similarly hold poor documentation responsible for team decisions becoming invisible and inadvertently causing hidden assumptions [34, 40, 43, 46, 76, 114]. Hopkins and Booth called model and data versioning in small companies as desired but "elusive" [40].

Recommendations. Many conflicts relate to boundaries of responsibility (especially for engineering responsibilities) and to different expectations by team members with different backgrounds. Better teams tend to define processes, responsibilities, and boundaries more carefully (📅), document APIs at collaboration points between teams (📄), and recruit dedicated engineering support for model deployment (⚙), but also establish a team culture with mutual understanding and exchange (👥). Big tech companies usually have more established processes and clearer responsibility assignments than smaller organizations and startups that often follow ad-hoc processes or figure out responsibilities as they go.

The need for engineering skills for ML projects has frequently been discussed [5, 67, 87, 90, 96, 112, 116], but our interviewees differ widely in whether all data scientists should have substantial engineering responsibilities or whether engineers should support data scientists so that they can focus on their core expertise (⚙). Especially interviewees from big tech emphasized that they expect engineering skills from all data science hires (P28a). Others emphasized that recruiting software engineers and operations staff with basic data-science knowledge can help at many communication and integration tasks, such as converting experimental ML code for deployment (P2a, P3b), fostering communication (P3c, P25a), and monitoring models in production (P5b). Generally, *siloing data scientists is widely recognized as problematic* and many interviewees suggest practices for improving communication (👥), such as training sessions for establishing common terminology (P11a, P17a, P22a, P22c, P23a), weekly all-hands meetings to present all tasks and synchronize (P2a, P3c, P6b, P11a), and proactive communication to broadcast upcoming changes in data or infrastructure (P11a, P14a, P14b). This mirrors suggestions to invest in interdisciplinary training [5, 48, 49, 69, 76, 112] and proactive communication [54].

7.3 Quality Assurance for Model and Product

During development and integration, questions of responsibility for quality assurance frequently arise, often requiring coordination and collaboration between multiple teams. This includes evaluating components individually (including the model) as well as their integration and the whole system, often including evaluating and monitoring the system *online* (in production).

Model adequacy goals are difficult to establish (📄, 👥). Offline accuracy evaluation of models is almost always performed by the model team responsible for building the model, though often they have difficulty deciding locally when the model is good enough (P1a, P3a, P5a, P6a, P7a, P15b, P16b, P23a) [34, 44]. As discussed in Sec. 5 and Sec. 6, model team members often receive little guidance on model adequacy criteria and are unsure about the actual distribution of production data. They also voice concerns about establishing ground truth, for example, needing to support data for different clients, and hence not being able to establish (offline) measures for model quality (P1b, P16b, P18a, P28a). As quality requirements beyond accuracy are rarely provided for models, model teams usually do not feel responsible for testing latency, memory consumption, or fairness (P2a, P3c, P4a, P5a, P6b, P7a, P14a, P15b, P20b). Whereas literature discussed challenges in measuring business impact of a model [10, 14, 43, 49] and balancing business goals with model goals [73], interviewed data scientists were concerned about this only with regards to convincing clients, managers or product teams to provide resources (P7a-b, P10a, P26a, P27a).

Limited confidence without transparent model evaluation (📄). Participants in several organizations report that model teams do not prioritize model evaluation and have no systematic evaluation strategy (especially if they do not have established adequacy criteria they try to meet), performing occasional "ad-hoc inspections" instead (P2a, P15b, P16b, P18b, P19b, P20b, P21b, P22a, P22b). Without transparency about their test processes and test results, other teams voiced reduced confidence in the model, leading to skepticism to adopt the model (P7a, P10a, P21b, P22a).

Unclear responsibilities for system testing (📅). Teams often struggle with testing the entire product after integrating ML and non-ML components. Model teams frequently explicitly mentioned that they assume no responsibility for product quality (including integration testing and testing in production) and have not been involved in planning for system testing, but that their responsibilities end with delivering a model evaluated for accuracy (P3a, P14a, P15b, P25a, P26a). However, in several organizations, product teams also did not plan for testing the entire system with the model(s) and, at most, conducted system testing in an ad-hoc way (P2a, P6a, P16a, P18a, P22a). Recent literature has reported a similar lack of focus on system testing in product teams [13, 114], mirroring also a focus in academic research on testing models rather than testing the entire system [10, 20]. Interestingly, some established software development organizations delegated testing to an existing separate quality assurance team with no process or experience testing ML products (P2a, P8a, P16a, P18b, P19a).

Planning for online testing and monitoring is rare (📅, ⚙, 👥). Due to possible training-serving skew and data drift, literature emphasizes the need for online evaluation [4, 10, 13, 14, 23, 42, 44, 47, 51, 65, 87, 88, 90, 103]. With collected telemetry, one can usually approximate both product and model quality, monitor updates, and experiment in production [14]. Online testing usually requires coordination among multiple teams responsible for product, model, and operation. We observed that most organizations do *not* perform monitoring or online testing, as it is considered difficult, in addition to lack of standard process, automation, or even test awareness (P2a, P3a, P3b, P4a, P6b, P7a, P10a, P15b, P16b, P18b, P19b, 25a, P27a). Only 11 out of 28 organizations collected any telemetry; it is most established in big tech organizations. When to retrain models is often decided based on intuition or manual inspection, though many aspire to more automation (P1a, P3a, P3c, P5a, P10a, P22a, P25a, P27a). Responsibilities around online evaluation are often neither planned nor assigned upfront as part of the project.

Most model teams are aware of possible data drift, but many do not have any monitoring infrastructure for detecting and managing drift in production. If telemetry is collected, it is the responsibility of the product or operations team and it is not always accessible to the model team. Four participants report that they rely on manual feedback about problems from the product team (P1a, P3a, P4a, P10a). At the same time, others report that product and operation teams do not necessarily have sufficient data science knowledge to provide meaningful feedback (P3a, P3b, P5b, P18b, P22a) [82].

Recommendations. Quality assurance involves multiple teams and benefits from explicit planning and making it a high priority (📅). While the product team should likely take responsibility for product quality and system testing, such testing often involves building monitoring and experimentation infrastructure (⚙), which requires planning and coordination with teams responsible for model development, deployment, and operation (if separate) to identify the right measures. Model teams benefit from receiving feedback on their model from production systems, but such support needs to be planned explicitly, with corresponding engineering effort assigned and budgeted, even in organizations following a model-first trajectory. We suspect that education about benefits of testing in production and common infrastructure (often under the label DevOps/MLOps [59]) can increase buy-in from all involved teams (👥). Organizations that have established monitoring and experimentation infrastructure strongly endorse it (P5a, P25a, P26a, P28a).

Defining clear quality requirements for model and product can help all teams to focus their quality assurance activities (cf. Sec. 5; 📄). Even when it is challenging to define adequacy criteria upfront, teams can together develop a quality assurance plan for model and product. Participants and literature emphasized the importance of human feedback to evaluate model predictions (P11a, P14a) [88], which requires planning to collect such feedback (📅). System and usability testing may similarly require planning for user studies with prototypes and shadow deployment [89, 100, 109].

8 DISCUSSION AND CONCLUSIONS

Through our interviews we identified three central collaboration points where organizations building ML-enabled systems face substantial challenges: (1) requirements and project planning, (2) training data, and (3) product-model integration. Other collaboration points surfaced, but were mentioned far less frequently (e.g., interaction with legal experts and operators), did not relate to problems between multiple disciplines (e.g., data scientists documenting their work for other data scientists), or mirrored conventional collaboration in software projects (e.g., many interviewees wanted to talk about unstable ML libraries and challenges interacting with teams building and maintaining such libraries, though the challenges largely mirrored those of library evolution generally [16, 31]).

Data scientists and software engineers are certainly not the first to realize that interdisciplinary collaborations are challenging and fraught with communication and cultural problems [21], yet it seems that many organizations building ML-enabled systems pay little attention to fostering better interdisciplinary collaboration.

Organizations differ widely in their structures and practices, and some organizations have found strategies that work for them (see recommendation sections). Yet, we find that most organizations do not deliberately plan their structures and practices and have little insight into available choices and their tradeoffs. We hope that this work can (1) encourage more deliberation about organization and process at key collaboration points, and (2) serve as a starting point for cataloging and promoting best practices.

Beyond the specific challenges discussed throughout this paper, we see four broad themes that benefit from more attention both in engineering practice and in research:

👥 **Communication:** Many issues are rooted in miscommunication between participants with different backgrounds. To facilitate interdisciplinary collaboration, education is key, including ML literacy for software engineers and managers (and even customers) but also training data scientists to understand software engineering concerns. The idea of T-shaped professionals [102] (deep expertise in one area, broad knowledge of others) can provide guidance for hiring and training.

📄 **Documentation:** Clearly documenting expectations between teams is important. Traditional interface documentation familiar to software engineers may be a starting point, but practices for documenting model requirements (Sec. 5.2), data expectations (Sec. 6.2), and assured model qualities (Sec. 7.3) are not well established. Recent suggestions like model cards [64], and FactSheets [8] are a good starting point for encouraging better, more standardized documentation of ML components. Given the interdisciplinary nature at these collaboration points, such documentation must be understood by all involved – theories of *boundary objects* [2] may help to develop better interface description mechanisms.

⚙ **Engineering:** With attention focused on ML innovations, many organizations seem to underestimate the engineering effort required to turn a model into a product to be operated and maintained reliably. Arguably adopting machine learning increases software complexity [48, 69, 87] and makes engineering practices such as data quality checks, deployment automation, and testing in production even more important. Project managers should ensure that the ML and the non-ML parts of the project have sufficient engineering capabilities and foster product and operations thinking from the start.

📅 **Process:** Finally, machine learning with its more science-like process challenges traditional software process life cycles. It seems clear that product requirements cannot be established without involving data scientists for model prototyping, and often it may be advisable to adopt a model-first trajectory to reduce risk. But while a focus on the product and overall process may cause delays, neglecting it entirely invites the kind of problems reported by our participants. Whether it may look more like the spiral model or agile [22], more research into integrated process life cycles for ML-enabled systems (covering software engineering and data science) is needed.

Acknowledgements. Kästner's and Nahar's work was supported in part by NSF grants NSF award 1813598 and 2131477. Zhou's work was supported in part by Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN2021-03538. Lewis' work was funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University (CMU) for the operation of the Software Engineering Institute (SEI), a federally funded research and development center. We would thank all our interview participants (K M Jawadur Rahman, Miguel Jette, and anonymous others) and the people who helped us connect with them.

REFERENCES

[1] Aho, T., Sievi-Korte, O., Kilamo, T., Yaman, S. and Mikkonen, T., 2020. Demystifying data science projects: A look on the people and process of data science today. *In Proc. Int'l Conf. Product-Focused Software Process Improvement*, 153-167.

[2] Akkerman, S.F. and Bakker, A. 2011. Boundary Crossing and Boundary Objects. *Review of educational research.* 81, 2, 132–169.

[3] Akkiraju, R., Sinha, V., Xu, A., Mahmud, J., Gundecha, P., Liu, Z., Liu, X. and Schumacher, J. 2020. Characterizing Machine Learning Processes: A Maturity Framework. *Business Process Management*, 17–31.

[4] Ameisen, E. 2020. *Building Machine Learning Powered Applications: Going from Idea to Product.* O'Reilly Media, Inc.

[5] Amershi, S. et al. 2019. Software Engineering for Machine Learning: A Case Study. *In Proc. of 41st Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291–300.

[6] Amershi, S., Chickering, M., Drucker, S.M., Lee, B., Simard, P. and Suh, J. 2015. ModelTracker: Redesigning Performance Analysis Tools for Machine Learning. *In Proc. of 33rd Conf. on Human Factors in Computing Systems*, 337–346.

[7] Amershi, S. et al. 2019. Guidelines for Human-AI Interaction. *In Proc. of CHI Conf. on Human Factors in Computing Systems*, 1–13.

[8] Arnold, M. et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development*, 63.

[9] Arpteg, A., Brinne, B., Crnkovic-Friis, L. and Bosch, J. 2018. Software Engineering Challenges of Deep Learning. *In Proc. Euromicro Conf. Software Engineering and Advanced Applications (SEAA)*, 50–59.

[10] Ashmore, R., Calinescu, R. and Paterson, C. 2021. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. *ACM Computing Surveys (CSUR)*, 54 (5): 1-39.

[11] Bass, L., Clements, P. and Kazman, R. 1998. *Software Architecture in Practice.* Addison-Wesley Longman Publishing Co., Inc.

[12] Bass, M., Herbsleb, J.D. and Lescher, C. 2009. A Coordination Risk Analysis Method for Multi-site Projects: Experience Report. *In Proc. Int'l Conf. Global Software Engineering*, 31–40.

[13] Baylor, D., Breck, E., Cheng, H.T., Fiedel, N., Foo, C.Y. et al. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *In Proc. Int'l Conf. Knowledge Discovery and Data Mining*, 1387-1395.

[14] Bernardi, L., Mavridis, T. and Estevez, P. 2019. 150 successful machine learning models. *In Proc. Int'l Conf. Knowledge Discovery & Data Mining*, 1743-1751.

[15] Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J.M.F. and Eckersley, P. 2020. Explainable machine learning in deployment. *In Proc. of Conf. on Fairness, Accountability, and Transparency*, 648–657.

[16] Bogart, C., Kästner, C., Herbsleb, J. and Thung, F. 2021. When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. *ACM Transactions on Software Engineering and Methodology.* 30, 4, 1–56.

[17] Borg, M. et al. 2019. Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. *Journal of Automotive Software Engineering.* 1, 1, 1–9.

[18] Bosch, J., Olsson, H.H. and Crnkovic, I. 2021. Engineering AI Systems: A Research Agenda. *Artificial Intelligence Paradigms for Smart Cyber-Physical Systems.* IGI Global. 1–19.

[19] Boujut, J.-F. and Blanco, E. 2003. Intermediary Objects as a Means to Foster Co-operation in Engineering Design. *Computer Supported Cooperative Work (CSCW).* 12, 2, 205–219.

[20] Braiek, H.B. and Khomh, F. 2020. On testing machine learning programs. *Journal of Systems and Software.* 164, 110542.

[21] Brandstädter, S. and Sonntag, K. 2016. Interdisciplinary Collaboration. *Advances in Ergonomic Design of Systems, Products and Processes*, 395–409.

[22] Braude, Eric J and Bernstein, Michael E. 2011. *Software Engineering: Modern Approaches 2nd Edition.* Wiley. *ISBN-13: 978-0471692089.*

[23] Breck, E., Cai, S., Nielsen, E., Salib, M. and Sculley, D. 2017. The ML test score: A rubric for ML production readiness and technical debt reduction. *In Proc. of Int'l Conf. on Big Data (Big Data)*, 1123–1132.

[24] Brown, G.F.C. 1995. *Factors that facilitate or inhibit interdisciplinary collaboration within a professional bureaucracy.* University of Arkansas.

[25] Cai, C.J., Winter, S., Steiner, D., Wilcox, L. and Terry, M. 2019. "hello AI": Uncovering the onboarding needs of medical practitioners for human-AI collaborative decision-making. *In Proc. Human-Computer Interaction.* 3, CSCW, 1–24.

[26] Cataldo, M. et al. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. *In Proc. Conf. Computer Supported Cooperative Work (CSCW)*, 353–362.

[27] Chattopadhyay, S., Prasad, I., Henley, A.Z., Sarma, A. and Barik, T. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. *In Proc. Conf. Human Factors in Computing Systems (CHI)*, 1–12.

[28] Cheng, D., Cao, C., Xu, C. and Ma, X. 2018. Manifesting Bugs in Machine Learning Code: An Explorative Study with Mutation Testing. *In Proc. Int'l Conf. Software Quality, Reliability and Security (QRS)*, 313–324.

[29] Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T. and Liu, X. 2020. Understanding Challenges in Deploying Deep Learning Based Software: An Empirical Study. *arXiv 2005.00760.*

[30] Conway, M.E. 1968. How Do Committees Invent? *Datamation.* 14, 4, 28–31.

[31] Cossette, B.E. and Walker, R.J. 2012. Seeking the Ground Truth: A Retroactive Study on the Evolution and Migration of Software Libraries. *In Proc. Int'l Symposium Foundations of Software Engineering (FSE)*, 1–11.

[32] Curtis, B., Krasner, H. and Iscoe, N. 1988. A field study of the software design process for large systems. *Communications of the ACM.* 31, 11, 1268–1287.

[33] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *In Proc. Conf. Computer Supported Cooperative Work (CSCW)*, 1277–1286.

[34] Haakman, M., Cruz, L., Huijgens, H. and van Deursen, A. 2021. AI Lifecycle Models Need To Be Revised. An exploratory study in Fintech. *Empirical Software Engineering.* 26, 5, 1–29.

[35] Haakman, M., Cruz, L., Huijgens, H. and van Deursen, A. 2020. AI Lifecycle Models Need To Be Revised. An Exploratory Study in Fintech. *arXiv 2010.02716.*

[36] Harsh, S. 2011. Purposeful Sampling in Qualitative Research Synthesis. *Qualitative Research Journal.* 11, 2, 63–75.

[37] Head, A., Hohman, F., Barik, T., Drucker, S.M. and DeLine, R. 2019. Managing messes in computational notebooks. *In Proc. Conf. Human Factors in Computing Systems (CHI)*, 1-12.

[38] Herbsleb, J.D. and Grinter, R.E. 1999. Splitting the Organization and Integrating the Code: Conway's Law Revisited. *In Proc. Int'l Conf. Software Engineering (ICSE)*, 85–95.

[39] Holstein, K. et al.. 2019. Improving Fairness in Machine Learning Systems: What Do Industry Practitioners Need? *In Proc. Conf. Human Factors in Computing (CHI) Systems*, 1–16.

[40] Hopkins, A. and Booth, S. 2021. Machine learning practices outside big tech: How resource constraints challenge responsible development. *In Proc. Conf. on AI, Ethics, and Society*, 134-145.

[41] Hukkelberg, I. and Rolland, K. 2020. Exploring Machine Learning in a Large Governmental Organization: An Information Infrastructure Perspective. *European Conf. on Information Systems*, 92.

[42] Hulten, G. 2019. *Building Intelligent Systems: A Guide to Machine Learning Engineering.* Apress.

[43] Humbatova, N. et al. 2020. Taxonomy of real faults in deep learning systems. *In Proc. Int'l Conf. on Software Engineering (ICSE)*, 1110-1121.

[44] Ishikawa, F. and Yoshioka, N. 2019. How do engineers perceive difficulties in engineering of machine-learning systems? - Questionnaire survey. *In Proc. Int'l Workshop on Conducting Empirical Studies in Industry (CESI) and Software Engineering Research and Industrial Practice (SER&IP)*, 2-9.

[45] Islam, M.J., Nguyen, H.A., Pan, R. and Rajan, H. 2019. What Do Developers Ask About ML Libraries? A Large-scale Study Using Stack Overflow. *arXiv 1906.11940.*

[46] Kandel, S., Paepcke, A., Hellerstein, J.M. and Heer, J. 2012. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics.* 18, 12, 2917–2926.

[47] Kang, D., Raghavan, D., Bailis, P. and Zaharia, M. 2020. Model Assertions for Monitoring and Improving ML Models. *In Proc. of Machine Learning and Systems, 2, 481-496.*

[48] Kästner, C. and Kang, E. 2020. Teaching Software Engineering for AI-Enabled Systems. *In Proc. Int'l Conf. Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 45–48.

[49] Kim, M., Zimmermann, T., DeLine, R. and Begel, A. 2018. Data Scientists in Software Teams: State of the Art and Challenges. *IEEE Transactions on Software Engineering.* 44, 11, 1024–1038.

[50] Kuwajima, H., Yasuoka, H. and Nakae, T. 2020. Engineering problems in machine learning systems. *Machine Learning*, 109, no 5, 1103-1126.

[51] Lakshmanan, V., Robinson, S. and Munn, M. 2020. *Machine Learning Design Patterns.* O'Reilly Media, Inc.

[52] Lewis, G.A., Bellomo, S. and Ozkaya, I. 2021. Characterizing and Detecting Mismatch in Machine-Learning-Enabled Systems. *In Proc. Workshop on AI Engineering-Software Engineering for AI (WAIN), 133-140.*

[53] Lewis, G. A., Ozkaya, I. and Xu X. 2021. Software Architecture Challenges for ML Systems. *In Proc. Int'l Conf. on Software Maintenance and Evolution*, 634-638.

[54] Li, P.L., Ko, A.J. and Begel, A. 2017. Cross-Disciplinary Perspectives on Collaborations with Software Engineers. *In Proc. Int'l Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2–8.

[55] Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H. and Crnkovic, I. 2019. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. *In Proc. Int'l Conf. Agile Software Development*, 227–243.

[56] Lwakatare, L.E., Raj, A., Crnkovic, I., Bosch, J. and Olsson, H.H. 2020. Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and software technology.* 127, 106368.

[57] Madaio, M.A. et al. 2020. Co-Designing Checklists to Understand Organizational Challenges and Opportunities around Fairness in AI. *In Proc. Conf. Human Factors in Computing Systems (CHI)*, 1–14.

[58] Mahanti, R. 2019. *Data Quality: Dimensions, Measurement, Strategy, Management, and Governance*. Quality Press.

[59] Mäkinen, S., Skogström, H., Laaksonen, E. and Mikkonen, T. 2021. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? *In Proc. Workshop on AI Engineering-Software Engineering for AI (WAIN)*, 109-112.

[60] Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M. and Wagner, S. 2021. Software Engineering for AI-Based Systems: A Survey. *arXiv 2105.01984*.

[61] Martinez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernandez Orallo, J., Kull, M., Lachiche, N., Ramirez Quintana, M.J. and Flach, P.A. 2021. CRISP-DM twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering*. 33, 8, 3048–3061.

[62] Meyer, B. 1997. *Object-Oriented Software Construction*. Prentice-Hall.

[63] Mistrík, I., Grundy, J., van der Hoek, A. and Whitehead, J. 2010. *Collaborative Software Engineering*. Springer.

[64] Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I.D. and Gebru, T. 2019. Model Cards for Model Reporting. *In Proc. Conf. Fairness, Accountability, and Transparency*, 220–229.

[65] Muiruri, D., Lwakatare, L. E., K Nurminen, J. and Mikkonen, T. 2021. Practices and Infrastructures for ML Systems–An Interview Study. *TechRxiv 16939192.v1*.

[66] Nahar, N., Zhou, S., Lewis, G. and Kästner, C. 2021. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. *arXiv 2110.10234*.

[67] O'Leary, K. and Uchida, M. 2020. Common problems with creating machine learning pipelines from existing code. *In Proc. Conf. Machine Learning and Systems (MLSys)*.

[68] Ovaska, P., Rossi, M. and Marttiin, P. 2003. Architecture as a coordination tool in multi-site software development. *Software Process Improvement and Practice*. 8, 4, 233–247.

[69] Ozkaya, I. 2020. What Is Really Different in Engineering AI-Enabled Systems? *IEEE Software*. 37, 4, 3–6.

[70] Panetta, K. 2020. Gartner Identifies the Top Strategic Technology Trends for 2021. *URL: https://www.gartner.com/smarterwithgartner/gartner-top-strategic-technology-trends-for-2021*.

[71] Park, S., Wang, A., Kawas, B., Vera Liao, Q., Piorkowski, D. and Danilevsky, M. 2021. Facilitating Knowledge Sharing from Domain Experts to Data Scientists for Building NLP Models. *In Proc. 26th Int'l Conf. on Intelligent User Interfaces*, 585-596.

[72] Parnas, D.L. 1972. On the Criteria to be used in Decomposing Systems into Modules. *Communications of the ACM*. 15, 12, 1053–1058.

[73] Passi, S., and Phoebe S. 2020. Making Data Science Systems Work. *Big Data & Society* 7 (2): 1-13.

[74] Patel, K., Fogarty, J., Landay, J.A. and Harrison, B. 2008. Investigating statistical machine learning as a tool for software development. *In Proc. Conf. Human Factors in Computing Systems (CHI)*, 667–676.

[75] Pimentel, J.F., Murta, L., Braganholo, V. and Freire, J. 2019. A large-scale study about quality and reproducibility of Jupyter notebooks. *In Proc. 16th Int'l Conf. on Mining Software Repositories (MSR)*, 507-517.

[76] Piorkowski, D. et al. 2021. How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study. *In Proc. ACM on Human-Computer Interaction*, 5, (CSCW1), 1-25.

[77] Polyzotis, N., Roy, S., Whang, S.E. and Zinkevich, M. 2018. Data Lifecycle Challenges in Production Machine Learning: A Survey. *SIGMOD Rec*. 47, 2, 17–28.

[78] Polyzotis, N., Roy, S., Whang, S.E. and Zinkevich, M. 2017. Data Management Challenges in Production Machine Learning. *In Proc. Int'l Conf. on Management of Data*, 1723–1726.

[79] Polyzotis, N., Zinkevich, M., Roy, S., Breck, E. and Whang, S. 2019. Data validation for machine learning. *In Proc. Machine Learning and Systems*, 334–347.

[80] Rahimi, M., Guo, J.L.C., Kokaly, S. and Chechik, M. 2019. Toward Requirements Specification for Machine-Learned Components. *In Proc. Int'l Requirements Engineering Conf. Workshops (REW)*, 241–244.

[81] Rakova, B., Yang, J., Cramer, H. and Chowdhury, R. 2021. Where Responsible AI meets Reality: Practitioner Perspectives on Enablers for Shifting Organizational Practices. *Proc. ACM Hum.-Comput. Interact*. 5, CSCW1, 1–23.

[82] Ré, C., Niu, F., Gudipati, P. and Srisuwananukorn, C. 2019. Overton: A data system for monitoring and improving machine-learned products. *arXiv 1909.05372*.

[83] Salay, R., Queiroz, R. and Czarnecki, K. 2017. An Analysis of ISO 26262: Using Machine Learning Safely in Automotive Software. *arXiv 1709.02435*.

[84] Sambasivan, N. et al. 2021. "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. *In Proc. Conf. on Human Factors in Computing Systems (CHI)*. 1–15.

[85] Sarma, A., Redmiles, D.F. and van der Hoek, A. 2012. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering*. 38, 4, 889–908.

[86] Schelter, S et al. 2018. Automating Large-scale Data Quality Verification. *Proc. VLDB Endowment Int'l Conf. Very Large Data Bases*. 11, 12, 1781–1794.

[87] Sculley, D. et al. 2015. Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems 28*. 2503–2511.

[88] Sculley, D., Otey, M.E., Pohl, M., Spitznagel, B., Hainsworth, J. and Zhou, Y. 2011. Detecting adversarial advertisements in the wild. *In Proc. Int'l Conf. Knowledge Discovery and Data Mining*, 274–282.

[89] Sendak, M.P. et al. 2020. Real-World Integration of a Sepsis Deep Learning Technology Into Routine Clinical Care: Implementation Study. *JMIR medical informatics*. 8, 7, e15182.

[90] Serban, A., van der Blom, K., Hoos, H. and Visser, J. 2020. Adoption and Effects of Software Engineering Best Practices in Machine Learning. *In Proc. Int'l Symposium on Empirical Software Engineering and Measurement*, 1–12.

[91] Seymoens, T., Ongenae, F. and Jacobs, A. 2018. A methodology to involve domain experts and machine learning techniques in the design of human-centered algorithms. *In Proc. IFIP Working Conf. Human Work Interaction Design*, 200-214.

[92] Shneiderman, B. 2020. Bridging the gap between ethics and practice. *ACM Transactions on Interactive Intelligent Systems*. 10, 4, 1–31.

[93] Siebert, J., Joeckel, L., Heidrich, J., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R. and Aoyama, M. 2020. Towards Guidelines for Assessing Qualities of Machine Learning Systems. *In Proc. Int'l Conf. on the Quality of Information and Communications Technology*, 17–31.

[94] Singh, G., Gehr, T., Püschel, M. and Vechev, M. 2019. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang*. 3, POPL, 1–30.

[95] Smith, D., Alshaikh, A., Bojan, R., Kak, A. and Manesh, M.M.G. 2014. Overcoming barriers to collaboration in an open source ecosystem. *Technology Innovation Management Review*. 4, 1.

[96] d. S. Nascimento, E. et al. 2019. Understanding Development Process of Machine Learning Systems: Challenges and Solutions. *In Proc. Int'l Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–6.

[97] de Souza, C.R.B. and Redmiles, D.F. 2008. An Empirical Study of Software Developers' Management of Dependencies and Changes. *In Proc. Int'l Conf. Software Engineering (ICSE)*, 241–250.

[98] Strauss, A. and Corbin, J. 1994. Grounded theory methodology: An overview. *Handbook of qualitative research*. N.K. Denzin, ed. 273–285.

[99] Strauss, A. and Corbin, J.M. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE Publications.

[100] Studer, S. et al. 2021. Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. *Machine Learning and Knowledge Extraction*, 3(2), 392-413.

[101] Tramèr, F. et al. 2017. FairTest: Discovering Unwarranted Associations in Data-Driven Applications. *In Proc. European Symposium on Security and Privacy (EuroS P)*, 401–416.

[102] Tranquillo, J. 2017. The T-Shaped Engineer. *Journal of Engineering Education Transformations*. 30, 4, 12–24.

[103] Vogelsang, A. and Borg, M. 2019. Requirements Engineering for Machine Learning: Perspectives from Data Scientists. *In Proc. Int'l Requirements Engineering Conf. Workshops (REW)*, 245–251.

[104] Wagstaff, K. 2012. Machine Learning that Matters. *arXiv 1206.4656*.

[105] Wang, A.Y., Mittal, A., Brooks, C. and Oney, S. 2019. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. *Proc. Human-Computer Interaction*. 3, CSCW, 39.

[106] Wan, Z., Xia, X., Lo, D. and Murphy, G.C. 2019. How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering*, 47(9), 1857-1871.

[107] Waterman, M., Noble, J. and Allan, G. 2015. How Much Up-Front? A Grounded theory of Agile Architecture. *In Proc. Int'l Conf. Software Engineering*, 347–357.

[108] Staff, V. B. 2019. Why do 87% of data science projects never make it into production? *URL: https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/*.

[109] Wiens, J., et al. 2019. Do no harm: A roadmap for responsible machine learning for health care. *Nature medicine*. 25, 9, 1337–1340.

[110] Xie, X., Ho, J.W.K., Murphy, C., Kaiser, G., Xu, B. and Chen, T.Y. 2011. Testing and Validating Machine Learning Classifiers by Metamorphic Testing. *Journal of Systems and Software*. 84, 4, 544–558.

[111] Yang, Q., Suh, J., Chen, N.-C. and Ramos, G. 2018. Grounding Interactive Machine Learning Tool Design in How Non-Experts Actually Build Models. *In Proc. Conf. Designing Interactive Systems*, 573–584.

[112] Yang, Q. The role of design in creating machine-learning-enhanced user experience. *In Proc. AAAI Spring Symposium Series*, 406-411.

[113] Yokoyama, H. 2019. Machine Learning System Architectural Pattern for Improving Operational Stability. *In Proc. Int'l Conf. on Software Architecture Companion (ICSA-C)*, 267–274.

[114] Zhang, A.X., Muller, M. and Wang, D. 2020. How do data science workers collaborate? Roles, workflows, and tools. *Proc. Human-Computer Interaction*. 4, CSCW1, 1–23.

[115] Zhou, S., Vasilescu, B. and Kästner, C. 2020. How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub. *In Proc. Int'l Conf. Software Engineering (ICSE)*, 445–456.

[116] Zinkevich, M. 2017. Rules of machine learning: Best practices for ML engineering. *URL: https://developers.google.com/machine-learning/guides/rules-of-ml*.