This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

How does Machine Learning Change Software Development Practices?

Zhiyuan Wan, Xin Xia, David Lo and Gail C. Murphy

Abstract—Adding an ability for a system to learn inherently adds uncertainty into the system. Given the rising popularity of incorporating machine learning into systems, we wondered how the addition alters software development practices. We performed a mixture of qualitative and quantitative studies with 14 interviewees and 342 survey respondents from 26 countries across four continents to elicit significant differences between the development of machine learning systems and the development of non-machine-learning systems. Our study uncovers significant differences in various aspects of software engineering (e.g., requirements, design, testing, and process) and work characteristics (e.g., skill variety, problem solving and task identity). Based on our findings, we highlight future research directions and provide recommendations for practitioners.

Index Terms—Software engineering, machine learning, practitioner, empirical study

1 Introduction

Machine learning (ML) has progressed dramatically over the past three decades, from a laboratory curiosity to a practical technology in widespread commercial use [19]. Within artificial intelligence, machine learning has emerged as the method of choice for developing useful software systems for computer vision, speech recognition, natural language processing, robot control, and other applications. Machine learning capabilities may be added to a system in several ways, including software systems with ML components and ML frameworks, tools and libraries that provide ML functionalities. A widespread trend has emerged: developing and deploying ML systems[1] is relatively fast and cheap, but maintaining them over time is difficult and expensive due to technical debt [19]. ML systems have all of the problems of non-ML software systems plus an additional set of ML specific issues. For instance, probabilistic modeling provides a framework for a machine to learn from observed data and infer models that can make predictions. Uncertainty plays a fundamental role in probabilistic modeling [14]: Observed data can be consistent with various models, and thus which model is appropriate given the data is uncertain. Predictions about future data and the future consequences of actions are uncertain as well. To tackle the ML specific issues, recent studies have put effort into building tools for testing [26], [30], [36], [39] and debugging [16], [27], [28] of machine learning code, and creating frameworks and environments to support development of ML systems [3], [6].

Despite these efforts, software practitioners still struggle to operationalize and standardize the software development practices of systems using ML[2]. Operationalization and standardization of software development practices are essential for cost-effective development of high-quality and reliable ML systems. How does machine learning change software development practices? To systematically explore the impact, we performed a mixture of qualitative and quantitative studies to investigate the differences in software development that arise from machine learning. We start with open-ended interviews with 14 software practitioners with experience in both ML and non-ML, who have an average of 7.4 years of software professional experience. Through the interviews, we qualitatively investigated the differences that were perceived by our interviewees and derived 80 candidate statements that describe the differences. We further improved the candidate statements via three focus group discussions and performed a survey with 342 software practitioners from 26 countries across four continents to quantitatively validate the differences that are uncovered in our interviews. The survey respondents work in various job roles, i.e., development (69%), testing (24%) and project management (7%). We investigated the following research questions:

RQ1. How does the incorporation of ML into a system impact software development practices?

- *Zhiyuan Wan is with the College of Computer Science and Technology, Zhejiang University, China and the Department of Computer Science, University of British Columbia, Canada. E-mail: wanzhiyuan@zju.edu.cn*
- *Xin Xia is with the Faculty of Information Technology, Monash University, Australia. E-mail: xin.xia@monash.edu*
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore. E-mail: davidlo@smu.edu.sg*
- *Gail C. Murphy is with the Department of Computer Science, University of British Columbia, Canada. E-mail: murphy@cs.ubc.ca*
- *Xin Xia is the corresponding author.*

Manuscript received ; revised

1. In this paper, unless otherwise mentioned, we use ML systems to refer to either software frameworks, tools, and libraries that provide ML functionalities, or software systems that include ML components.

2. https://twitter.com/AndrewYNg/status/1080886439380869122

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

Is developing ML systems different from developing non-ML systems? How does it differ? If developing ML systems is indeed different from non-ML software development, past software engineering research may need to be expanded to better address the unique challenges of developing ML systems; previous tools and practices may become inapplicable to the development of ML systems; software engineering educators may need to teach different skills for the development of ML systems. Our study found several statistically significant differences in software engineering practices between ML and non-ML development:

- **Requirements**: Collecting requirements in the development of ML systems involves more preliminary experiments, and creates a need for the predictable degradation in the performance.
- **Design**: Detailed design of ML systems is more time-consuming and tends to be conducted in an intensively iterative way.
- **Testing and Quality**: Collecting a testing dataset requires more effort for ML development; Good performance[3] during testing cannot guarantee the performance of ML systems in production.
- **Process and Management**: The availability of data limits the capability of ML systems; Data processing is more important to the success of the whole process.

RQ2. How do the work characteristics from applied psychology, like skill variety, job complexity and problem solving, change when incorporating ML into a system?

How does the context of software development (e.g., skill variety, job complexity and problem solving) change, when practitioners involve ML in their software development practices? Our study identified several statistically significant differences in work characteristics between ML and non-ML development:

- **Skill Variety**: ML development intensively requires knowledge in math, information theory, and statistics.
- **Job Complexity and Problem Solving**: ML practitioners have a less clear roadmap for building systems.
- **Task Identity**: It is much harder to make an accurate plan for the tasks for ML development.
- **Interaction**: ML practitioners tend to communicate less frequently with clients.

Based on the findings, we present the causes behind the identified differences as discussed by our interviewees - the uncertainty in requirements and algorithms, and the vital role of data. We also provide practical lessons about the roles of preliminary experiments, reproducibility and performance reporting, and highlight several research avenues such as continuous performance measurement and debugging.

3. In this paper, unless otherwise mentioned, we use performance to refer to model performance.

This paper makes the following contributions:

- We performed a mixture of qualitative and quantitative studies to investigate the differences in software practices and practitioners' work due to the impact of machine learning;
- We provided practical implications for researchers and outlined future avenues of research.

The remainder of the paper is structured as follows. Section 2 briefly describes the processes and concepts regarding ML development. In Section 3, we describe the methodology of our study in detail. In Section 4, we present the results of our study. In Section 5, we discuss the implications of our results as well as any threats to the validity of our findings. In Section 6, we briefly review related work. Section 7 draws conclusions and outlines avenues for future work.

2 Background

The development of machine learning systems is a multi-faceted and complex task. Various forms of processes of ML development have been proposed [2], [11], [12], [35]. These processes share several common essential steps: context understanding, data curation, data modeling, and production and monitoring.

In the *context understanding* step, ML practitioners identify areas of business that could benefit from machine learning and the available data. ML practitioners would communicate with stakeholders about what machine learning is capable and not capable of to manage expectations. Most importantly, ML practitioners frame and scope the development tasks by conducting preliminary experiments in a particular application context.

The *data curation* step includes *data collection* from different sources, *data preprocessing*, and training, validation and test *dataset creation*. Since data often come from different sources, ML practitioners should stitch together data, and deal with missing or corrupted data through *data preprocessing*. To create an appropriate dataset for supervised learning techniques, *data labeling* is required to assign ground truth labels to each record.

The *data modeling* step includes *feature engineering*, *model training*, and *model evaluation*. *Feature engineering* refers to the activities that transform the given data into a form which is easier to interpret, including feature extraction and selection for machine learning models. During *model training*, ML practitioners choose, train, tune machine learning models using the chosen features. Model tuning includes adjusting parameters and identifying potential issues in the current model or the previous steps. In *model evaluation*, practitioners evaluate the output model on the test dataset using pre-defined evaluation measures.

During the *production and monitoring* step, ML practitioners export the model into a pre-defined format and usually create an API or Web application with the model as an endpoint. ML practitioners also plan for retraining the model with updated data. The model performance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

is continuously monitored for errors or unexpected consequences, and input data are monitored to identify if they change with time in a way that would invalidate the model.

We use the process above of ML development and related terminology as the vocabulary for discussions in this work.

3 Methodology

Our research methodology followed a mixed qualitative and quantitative approach as depicted in Fig. 1. We collected data from different sources[4]: (1) We interviewed 14 software practitioners with experience in both ML development and non-ML development; (2) We derived a list of 80 candidate statements from the results of interviews, and conducted three focus group discussions to reduce our list to 31 final statements for our survey; (3) We surveyed 342 respondents, which we describe below. To preserve the anonymity of participants, we anonymized all items that constitute of Personally Identifiable Information (PII) before analyzing the data, and further considered aliases as PII throughout our study (e.g., refer to the interviewees as P1 - P14).

3.1 Interviews

3.1.1 Protocol

The first author conducted a series of face-to-face interviews with 14 software practitioners with experience in both ML development and non-ML development. Each interview took 30-45 minutes. According to Guest et al. [15], conducting 12 to 15 interviews of a homogeneous group is adequate to reach saturation. We observed a saturation when our interviews were drawing to a close. The interviews were semi-structured and made use of *an interview guide*[5]. The guide contains general groupings of topics and questions, rather than a pre-determined specific set and order of questions.

The interview comprised four parts. In the first part, we asked some demographic questions about the experience of the interviewees in both ML development and non-ML development. We covered various aspects including programming, design, project management, and testing.

In the second part, we asked an open-ended question about what differences the interviewee noticed between ML development versus non-ML development. The purpose of this part was to allow the interviewees to speak freely about differences without the interviewer biasing their responses.

In the third and fourth part, we presented interviewees with two lists of topics and asked them to discuss the topics that they have not explicitly mentioned. One of the list comes from the Guide to the Software Engineering Body of Knowledge (SWEBOK) [7], which consists of 10 knowledge areas, e.g., *software design* and *software testing*. The other list comes from general work characteristics [18] in applied psychology, which consists of 21 work characteristics, e.g., *skill variety* and *problem solving*. We chose SWEBOK to ensure that software engineering topics were comprehensively discussed, and general work characteristics to ensure that we covered a breadth of potential differences. In the third part, interviewees were asked to choose three topics from the two lists to discuss. In the fourth part, interviewer selected three topics from the two lists that had been discussed the least in previous interviews, to ensure coverage of the topics.

At the end of each interview, we thanked the interviewee and briefly informed him/her of our next plans.

During the interviews, each interviewee talked about a median of 6 topics where he/she shared his/her perceived difference between ML development and non-ML software development (min: 1, max: 12, mean: 6.6, sd: 3.2). The topics mentioned by the interviewees include: *SWEBOK: Requirements* (9 interviewees), *SWEBOK: Design* (6 interviewees), *SWEBOK: Construction* (10 interviewees), *SWEBOK: Tools* (7 interviewees), *SWEBOK: Testing* (9 interviewees), *SWEBOK: Quality* (5 interviewees), *SWEBOK: Maintenance* (4 interviewees), *SWEBOK: Process* (8 interviewees), *SWEBOK: Configuration Management* (3 interviewees), *Work: Skill Variety* (10 interviewees), *Work: Job Complexity* (5 interviewees), *Work: Problem Solving* (4 interviewees), *Work: Task Identify* (7 interviewees), *Work: Autonomy* (1 interviewees), *Work: Interdependence* (4 interviewees), and *Work: Interaction Outside the Organization* (1 interviewees).

3.1.2 Participant Selection

We recruited full-time employees with experience in both ML systems and non-ML systems from three IT companies based in Hangzhou, China, namely Alibaba, Bangsun[6], and Hengtian[7]. Bangsun is a technology provider which has more than 400 employees and develops real-time risk control systems for the financial sector and anti-fraud products. Hengtian is an outsourcing company which has more than 2,000 employees and focuses on outsourcing projects from US and European corporations (e.g., State Street Bank, Cisco, and Reuters). Interviewees were recruited by emailing our contact in each company, who was then responsible for disseminating news of our study to their colleagues. Volunteers would inform us if they were willing to participate in the study with no compensation. With this approach, 14 volunteers contacted us with varied experience in years. In the remainder of the paper, we denote these 14 interviewees as P1 to P14. These 14 interviewees have an average of 7.6 years of professional experience (min:

4. The interviews, focus group and survey were approved by the relevant institutional review board (IRB). Participants were instructed that we wanted their opinions; privacy and sensitive resources were not explicitly mentioned.

5. Interview Guide Online: https://drive.google.com/file/d/1ZOXwbSKY6zPnuOEzGlzFMJ3DIERYD8YG

6. https://www.bsfit.com.cn

7. http://www.hengtiansoft.com/?lang=en

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

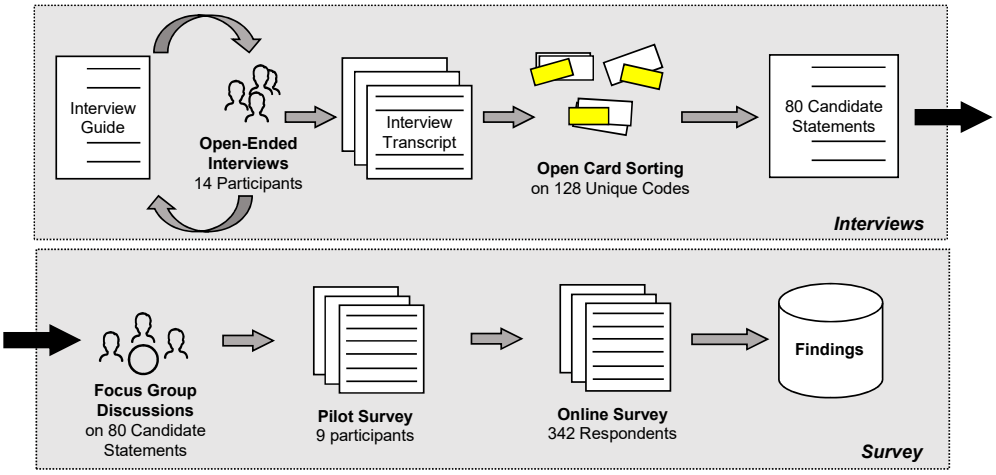

Fig. 1: Research methodology.

TABLE 1: Number of interviewees with "extensive" experience in a particular role.

Role	Machine Learning	non-Machine-Learning
Programming	5	6
Design	5	3
Management	2	2
Testing	2	3

3, max: 16, median: 6, sd: 4), including 2.4 years in ML system development (min: 1, max: 5, median: 2, sd: 1.6) and 5.2 years in non-ML software development (min: 2, max: 11, median: 4.5, sd: 2.6). Table 1 summarizes the number of interviewees who perceived themselves with "extensive" experience (in comparison to "none" and "some" experience) in a particular role.

3.1.3 Data Analysis

We conducted a thematic analysis [8] to process the recorded interviews by following the steps below:

Transcribing and Coding. After the last interview was completed, we transcribed the recordings of the interviews, and developed a thorough understanding through reviewing the transcripts. The first author read the transcripts and coded the interviews using NVivo qualitative analysis software [1]. To ensure the quality of codes, the second author verified initial codes created by the first author and provided suggestions for improvement. After incorporating these suggestions, we generated a total of 295 cards that contain the codes - 15 to 27 cards for each coded interview. After merging the codes with the same words or meanings, we have a total of 128 unique codes. We noticed that when our interviews were drawing to a close, the collected codes from interview transcripts reached a saturation. New codes did not appear anymore; the list of codes was considered stable.

Open Card Sorting. Two of the authors then separately analyzed the codes and sorted the generated cards into potential themes for thematic similarity (as illustrated in LaToza et al.'s study [24]). The themes that emerged during the sorting were not chosen beforehand. We then use the Cohen's Kappa measure [10] to examine the agreement between the two labelers. The overall Kappa value between the two labelers is 0.78, which indicates substantial agreement between the labelers. After completing the labeling process, the two labelers discussed their disagreements to reach a common decision. To reduce bias from two of the authors sorting the cards to form initial themes, they both reviewed and agreed on the final set of themes. Finally, we derived 80 candidate statements that describe the differences.

3.2 Focus Groups

To focus the survey and keep it to a manageable size, we wanted to hone in on the statement that are most likely to differ when ML is incorporated into a software system. To determine which of the 80 candidate statements had this characteristic, the first author conducted three focus group sessions. Each focus group session lasted for 1.5 to 2 hours and involved 3 participants. The 9 participants are professionals with experience in both ML and non-ML development from various IT companies in China (e.g., Baidu, Alibaba and Huawei). They were informed about the purpose of our study and gave their consent to use the focus group results for research purposes.

During the focus group sessions, the first author went through the 80 candidate statements, and asked the following question "is the statement more true for ML development, in comparison with non-ML development". Based on the feedback, we removed 7 statements in which the participants did not understand the difference or did not think there was a difference for ML vs. non-ML development. In addition, we removed 42 statements in which over half of our focus group participants perceived no obvious difference between ML and non-ML development. In the end, we identified a list of 31 statements.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

3.3 Survey

3.3.1 Protocol

The survey aims to quantify the differences between ML and non-ML software development expressed by interviewees over a wide range of software practitioners. We followed Kitchenham and Pfleeger's guidelines for personal opinion surveys [23] and used an anonymous survey to increase response rates [37]. A respondent has the option to specify that he/she prefers not to answer or does not understand the description of a particular question. We include this option to reduce the possibility of respondents providing arbitrary answers.

Recruitment of Respondents. The participants of the survey were informed about the purpose of our study and gave their consent to use the survey results for research purposes.

To recruit respondents from both ML and non-ML populations, we spread the survey broadly to a wide range of companies from various locations around the world. To get a sufficient number of respondents from diverse backgrounds, we followed a multi-pronged strategy to recruit respondents:

- We contacted professionals from various countries and IT companies and asked their help to disseminate our survey within their organizations. We sent emails to our contacts in Amazon, Alibaba, Baidu, Google, Hengtian, IBM, Intel, IGS, Kodak, Lenovo, Microsoft, Morgan Stanley, and other companies from various locations around the world, encouraging them to complete the survey and disseminate it to some of their colleagues. By following this strategy, we aimed to recruit respondents working in the industry from diverse organizations.
- We sent an email with a link to the survey to 1,831 practitioners that contributed to 18 highest-rated machine learning repositories hosted on GitHub (e.g., TensorFlow and PyTorch) and solicited their participation. By sending to GitHub contributors to machine learning repositories, we aimed to recruit respondents who are open source practitioners in addition to professionals working in the industry. We chose this set of potential respondents to collect responses from ML practitioners for contrast; if ML respondents provide significantly different responses than non-ML respondents, this provides quantitative evidence to establish a difference between ML and non-ML development. Moreover, the reason for choosing the 18 high-rated machine learning repositories was that the contributors would potentially be two types: practitioners of ML framework/tool/library (ML FTL) and practitioners of ML application[8] (ML App). We were unsure whether high variances in software differences would overwhelm ML versus non-ML differences. Out of these emails, eight emails received automatic replies notifying us of the absence of the receiver.

8. A software system with ML components.

No identifying information was required or gathered from our respondents.

3.3.2 Survey Design

We captured the following pieces of information in our survey (the complete questionnaire is available online as supplemental material[9]):

Demographics. We collected demographic information about the respondents to allow us to (1) filter respondents who may not understand our survey (i.e., respondents with less relevant job roles), (2) breakdown the results by groups (e.g., developers and testers; ML practitioners and non-ML practitioners). Specifically, we asked the question "*What best describes your **primary product area** that you currently work on*?", and provided options including (1) *ML framework/system/library*, (2) *ML application*, (3) *Non-ML framework/system/library*, (4) *Non-ML application*, and (5) *Other*. The respondents selected one item from the provided options as their primary product area. Based on their selections, we divided the survey respondents into 5 groups.

We received a total of 357 responses. We excluded ten responses made by respondents whose job roles are neither development, testing nor project management. Those respondents describe their job roles as a researcher (5), student (3) network infrastructure specialist (1), and university professor (1). We also excluded five responses made by respondents who selected *Other* as major product areas and specified their major product areas as: ecology (1), physics (1), or a combination of multiple product areas that do not seem oriented at the production of a commercially relevant software product (3). In the end, we had a set of 342 valid responses.

The 342 respondents reside in 26 countries across four continents as shown in Fig. 2. The top two countries in which the respondents reside are China and the United States. The number of years of professional experience of the respondents varied from 0.1 to 25 years, with an average of 4.3 years. Our survey respondents are distributed across different demographic groups (job roles and product areas) as shown in Fig. 3.

Practitioners' Perceptions. We provided the list of 31 final statements, and asked practitioners to respond to each statement on a 5-point Likert scale (*strongly disagree*, *disagree*, *neutral*, *agree*, *strongly disagree*). To focus the respondents' attention on a particular area in the survey, they were explicitly asked to rate each statement with respect to their experience with the *major product area* they specified.

We piloted the preliminary survey with a small set of practitioners who were different from our interviewees, focus-group participants and survey takers. We obtained feedback on (1) whether the length of the survey was appropriate, and (2) the clarity and understandability of the terms. We made minor modifications to the preliminary

9. Questionnaire Online: https://drive.google.com/file/d/124ttMmqSXglilEUuevAMP85jvP4uyVoc

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

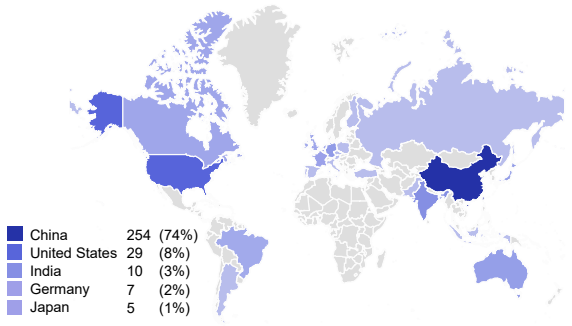

Fig. 2: Countries in which survey respondents reside. The darker the color is, the more respondents reside in that country. The legend presents the top 5 countries with most respondents.

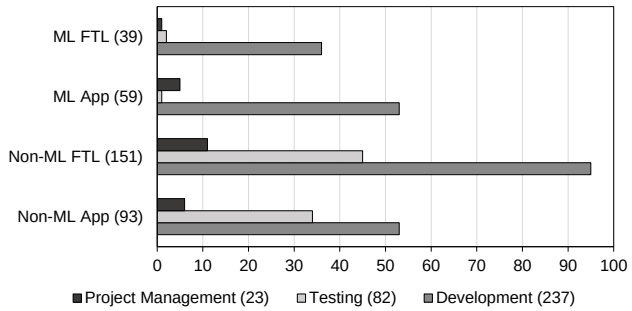

Fig. 3: Survey respondents demographics. The number indicates the count of each demographic group.

survey based on the received feedback and produced a final version. Note that the collected responses from the pilot survey are excluded from the presented results in this paper.

To support respondents from China, we translated our survey to Chinese before publishing the survey. We chose to make our survey available both in Chinese and English because Chinese is the most spoken language and English is an international lingua franca. We expect that a large number of our survey recipients are fluent in one of these two languages. We carefully translated our survey to make sure there exists no ambiguity between English and Chinese terms in our survey. Also, we polished the translation by improving clarity and understandability according to the feedback from our pilot survey.

3.3.3 Data Analysis

We examined distributions of Likert responses for our participants and compared the distributions of different groups of participants using Wilcoxon rank-sum test, i.e., ML vs. non-ML, and ML framework/tool/library vs. ML application. We report the full results in Section 4.3; along the way in Section 4.1 and 4.2, we link interviewees' comments with survey responses by referring to survey statements like: [S1]. We number statements in the order in which they appeared in the survey, S1 through S31. We annotate each with whether they are statistically significant or not as follows:

- **[✔ S1]** Significant difference between ML development and non-ML development that *confirms* interviewees' responses and no significant difference between the development of ML framework/tool/library and development of ML software application;
- **[✔ ✔ S1]** Significant difference between ML development and non-ML development, and significant difference between the development of ML framework/tool/library and development of ML software application;
- **[S1]** No significant differences;
- **[✘ S1]** Significant difference between ML development and non-ML development, but *opposite* of interviewees' responses.
- **[✘ ✔ S1]** Significant difference between ML development and non-ML development, but *opposite* of interviewees' responses; and significant difference between development of ML framework/tool/library and development of ML software application.
- **[✔ S1]** No significant difference between ML development and non-ML development; but significant difference between development of ML framework/tool/library and development of ML software application.

Other outcomes are theoretically possible but did not occur in our survey results.

4 Results

In this section, we report the results grouped based on the interview topics. We combined several topics into one when interviewees had little to say about a particular topic. In some cases, we have anonymized parts of quotes to maintain interviewees' privacy.

4.1 RQ1. Differences in Software Engineering Practices

4.1.1 Software Requirements

Nearly every interviewee made a strong statement about differences between the requirements of ML systems versus the requirements of non-ML systems. In essence, requirements of ML systems are generally data-driven - closely coupled with existing large-scale data of a particular application context.

Interviewees noted that requirements are more uncertain for ML systems than non-ML systems [S1]. As P9 noted, given "machine learning systems usually aim to improve or accelerate the decision-making process (of executives in an organization or a company)", rather than detailed functional descriptions, the requirements usually include a conceptual description about the goal after applying the machine learning systems. Since the requirements of machine learning systems are data-driven, different data would lead to different requirements. Even for the same data, as P1 and P6 suggested,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

a different understanding of the data and different application contexts would lead to different requirements. Nevertheless, prior knowledge about the data and application contexts bring determinism to a certain extent. P6 gave a specific example, suggesting that how prior knowledge helps to understand the data and application contexts:

> *There exists a kind of prior knowledge named "scenario prior knowledge". For instance, we know that the data imbalance problem occurs in the application of fraud detection. This is because good guys always account for a larger amount of people than bad guys. As we also know, in the field of online advertising, the conversion rate*[10] *usually seems low and there exists a limit.*

Instead of functional requirements in non-ML software systems, quantitative measures comprise the majority of requirements for ML systems. As P4 pointed out, distinct types of quantitative measures would be leveraged to define requirements, e.g., *accuracy*, *precision*, *recall*, *F measure* and *normalized discounted cumulative gain* (nDCG). These quantitative measurements could either come from the "captain's call" by business stakeholders (P6) or be collected by project managers through user studies (P5). As P4 put it, the target scores for quantitative measures could vary from one application to another. In some safety-critical domains, the accuracy of ML systems is of great importance. As a consequence, higher scores of quantitative measures are expected for safety considerations. P5 echoed this, saying

> *For online shopping recommendation systems, the quantitative measures are relatively not so restricted, and lower measures are tolerable.*

In contrast to non-ML systems, requirements for ML systems usually involve a large number of preliminary experiments [✔ S2]. As P6 noted, business stakeholders might suggest leveraging a number of emerging machine learning algorithms to solve their business problems. One of the consequences is that it requires the requirement specialists to have a strong technical background in machine learning. The other consequence is that requirement validation process involves a larger amount of preliminary experiments. Those preliminary experiments are conducted by software engineers and intend to validate and select machine learning algorithms among various candidates. As P8 explained,

> *Say A, B and C algorithms might be all suitable for a particular application context, but the performance closely depends on the actual data in practice. Requirements cannot be validated until preliminary experiments have been conducted.*

The requirement should consider the predictable degradation in the performance of ML systems [✔ S3]. As P6 noted, most of the ML systems might experience performance degradation after a period in production. P6 gave an example: In a fraud detection application, adversaries are always trying to find ways to evade detection strategies. As a result, two inherent requirements are expected for ML systems. First, ML systems are expected to be degradation-sensitive, i.e., be capable of perceiving performance degradation. Second, once a performance degradation occurs, ML system needs to have considerable capability to adapt to the degradation, either by feeding new data to the learning algorithm or training a brand new model by using new data.

As we will discuss in subsequent sections, data-driven and large-scale characteristics of ML systems have several consequences to the way they are developed, compared to non-ML systems.

4.1.2 Software Design

Interviewees repeatedly mentioned that the design of ML systems and non-ML software systems differently place emphasis in a few ways.

First, the high-level architectural design for ML systems is relatively fixed [✘ S4]. As P3 summarized, the architecture of ML systems typically consists of data collection, data cleaning, feature engineering, data modeling, execution, and deployment. In contrast, the architectural design for non-ML software systems is a more creative process, which implements various structural partitioning of software components and generates behavioral descriptions (e.g., activity diagrams and data flow diagrams) (P12). Due to the high volume of data, the distributed architectural style is widely preferred for ML systems. Distributed architectural style usually leads to complexity in architectural and detailed design.

Second, ML systems place less emphasis on low coupling in components than non-ML software systems [S5]. Although different components in ML systems have separate functionalities, they are highly coupled. For instance, the performance of data modeling is dependent on data processing. As P14 noted, "'garbage in and garbage out' - I would spend 40% of my time on data processing since I found that poor data processing could fail any potential effective [machine learning] models ... I divide the data processing into multiple steps and may use existing libraries for each step".

Third, detailed design is more flexible for ML systems than non-ML software systems [S6]. P1 noted that data modeling could contain tens to hundreds of candidates of machine learning algorithms, which indicates an ample search space. P6 echoed this, saying

> *Even for the same machine learning algorithm, various application contexts may introduce differences in the dimensions of data, and further lead to changes in machine learning models.*

As a consequence, the detailed design of ML systems would be time-consuming and conducted in an iterative way [✔ S7]. To design an effective model, software engineers tend to conduct a large number of experiments.

10. The probability that the user who sees the ad on his or her browser will take an action, i.e., the user will convert [25].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

4.1.3 Software Construction and Tools

Interviewees reported several differences between ML systems and non-ML software systems in terms of coding practice. First, the coding workload of ML systems is low compared to non-ML software systems [S8]. Instead of coding for implementing particular functionalities in non-ML software systems, coding in ML systems generally includes data processing (e.g., transformation, cleansing, and encoding), feature analysis (e.g., visualization and statistical testing), and data modeling (e.g., hyperparameters selection and model training). P14 pointed at the availability of useful frameworks and libraries for data processing and data modeling. These frameworks and libraries help developers accelerate the coding process. To achieve better performance, developers can extend these frameworks or libraries to be adapted for their own use. Second, there is little code reuse between and within ML systems, compared to non-ML software systems [S9]. One reason is that ML systems frequently have a significant emphasis on performance. However, the performance of ML systems highly depends on the data; data vary across different application contexts. Thus, project-specific performance tuning is necessary.

Debugging in non-ML software systems aims to locate and fix bugs in the code [S10]. Unlike non-ML software systems, debugging in ML systems aims to improve performance. The performance of ML systems generally cannot be aware or evaluated "until the last minute when the data model is finalized" (P13, P14). Efficiently finalizing a data model is playing an important role in the construction of ML systems. However, data modeling involves multiple iterative training rounds. Considering the high volume of data, each round of training may take a long time, days or weeks, if complete data are taken. It is infeasible to use complete data to train models for each round. Thus, several interviewees suggested a practical data modeling process (P4, P6, P13):

> *You need to build several training datasets of different sizes from small-scale to large-scale. You start with the small-scale dataset to train models. Till you achieve acceptable results, you move to a larger scale. Finally, all the way up, you would find a satisfactory model.*

Although this process improves the training efficiency, incomplete training data might risk introducing inaccuracy in intermediate results that may lead to bias in models.

Interviewees mentioned that ML systems and non-ML software systems differ in debugging practice. Debugging practice of non-ML software systems typically uses step-by-step program execution through breakpoints. For ML systems, especially deep learning software systems, "debugging aims to make it more straightforward to translate ideas from developer's head into code". P6 gave a specific example about "dynamic computational graph":

> *Previously, developers prefer to use PyTorch mainly because of its support for the dynamic computational graph. 'dynamic computation' means that the model is executed in the order you wrote it. Well, like I am building a neural network, I would add up layers one by one. Then each layer has some tradeoffs; for instance, I would like to add an operator layer implementing normalization. [If a debugging tool does not support dynamic computational graph,] I cannot evaluate if this addition is good or not until the neural network is compiled into a model and real data go in. The dynamic computational graph allows debugging on sample data immediately and helps me verify my idea quickly.*

Nevertheless, debugging on the dynamic computational graph has drawbacks. Once the data volume is extremely high, computation for each layer takes a long time to finish. This delays the construction of further layers. In addition, interviewees also mentioned that creativity appears to be important in debugging of ML systems [S11]. Part of the reason appears to be that because ML systems have an extensive search space for model tuning.

Interviews pointed at several differences in bugs between ML systems and non-ML software systems. First, ML systems do not have as many bugs that reside in the code as non-ML software (P11) [S12]. Instead, bugs are sometimes hidden in the data (P10). P1 gave a recent example that misusing training data and testing data with intensive overlap results in an incredibly good performance, but indicating a bug. As P4 suggested, "generalization of data models is also required to be taken care of". Second, ML systems have specific types of bugs when taking data into account. As P11 stated, the mismatch of data dimension order between two frameworks may cause bugs when integrating these two frameworks. Third, in contrast to non-ML software systems, a single failed case is hardly helping diagnose a bug in ML systems. As P13 explained, sometimes, developers of ML systems find bugs by just "staring at every line of their code and try to think why it would cause a bug".

4.1.4 Software Testing and Quality

Although software quality is important in both ML and non-ML systems, the practice of testing appears to differ significantly. One significant difference exists in the reproducibility of test results. In contrast to non-ML software systems, the testing results of ML systems is hard to reproduce because of a number of sources of randomness [S13]. As P8 explained:

> *The randomness [in ML systems] complicates testing. You have random data, random observation order, random initialization for your weights, random batches fed to your network, random optimizations in different versions of frameworks and libraries ... While you can seed the initialization, fixing the batches might come with a performance hit, as you would have to turn-off parallel batch generation which many frameworks do. There is also the approach to freeze or write-out the weights after just one iteration which solves the weight-initialization randomness.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

Another difference exists in the testing methods and resulting outputs. Testing practice in ML systems usually involves running an algorithm multiple times and gather a population of performance measurements [S14]. As P12 explained,

> *Testing practice for machine learning software mainly aims to verify the quantitative measures that indicate performance. However, machine learning algorithms are stochastic. Thus, we usually use k-fold cross-validation to do the testing.*

As P9 echoed, the testing outputs are expected to be a range rather than a single value.

The interviewees stated that test case generation for ML systems is more challenging, compared to non-ML systems. Automated testing tools are not used as frequently as non-ML systems [✘ ✔ S18]. The test dataset is essential to the quality of test cases. Collecting testing datasets is labor intensive [✔ ✔ S15]. If the application is for general use where correct answers are known to human users, labeling tasks could be outsourced to non-technical people outside the organization, as P5 noted. More details are needed for these automated methods or tools. However, biases may be introduced to test dataset through the methods or tools, and consequently, affect both performance and generalizability. As P8 put it,

> *Sometimes, we (developers) generate expected results for the test cases using the algorithms or models constructed by ourselves. Paradoxically, this may introduce bias because it is like we define acceptance criteria for our code.*

Moreover, generating reliable test oracle is sometimes infeasible for some ML systems. P6 gave a specific example in the anomaly detection application context:

> *Clients gave us a test dataset and told us the dataset contains labeled anomaly cases. However, we have no way to know how many anomaly cases exactly there are in the dataset, because some anomalies may not have been recognized and labeled in the dataset.*

Good testing results cannot guarantee the performance of ML systems in production [✔ S17]. The performance in production to a large extent depends on how similar the training dataset and the incoming data are (P6).

In ML systems, "too low" and "too high" scores for performance measures as testing results both indicate bugs [S16]. P1 gave a recent example of a junior developer who obtained an F1 score of 99% in his data model. In fact, after carefully going through the dataset, an extensive overlap was discovered between training and testing dataset. Some interviewees reported several specific tactics in testing ML systems (P13):

> *We can use a simple algorithm as the baseline, for example, a random algorithm. If the baseline performs quite well on our dataset, there might exist bugs in our dataset. If our algorithm performs worse than the baseline, there might be some bugs in our code.*

4.1.5 Software Maintenance and Configuration Management

Interviewees suggested that less effort may be required in the maintenance for ML systems than traditional software systems [S19]. One reason is that, different from non-ML software systems, ML systems run into predictable degradation in performance as time goes by (P4 and P7). To provide constantly robust results, ML systems should support "automatic" maintenance. Once performance degradation occurs, an ML system is designed to perceive the degradation and trains new data models in an online/offline way using the latest emerged data. As P6 suggested,

> *We sometimes define the so-called "health factors" or quantitative indicators, of the status of a machine learning system. They are associated with a specific application context. The indicators help machine learning system perceives its performance in the specific application context.*

Interviewees reported that configuration management for ML systems involves a larger amount of content compared to non-ML software [S20]. One reason is that machine learning models include not only code but also data, hyperparameters, and parameters. Developing ML systems involves rapid experimentation and iteration. The performance of models would vary accordingly. To find the optimal combination of these parts that achieve the best performance, configuration management is required to keep track of the varying models and associated tradeoffs, algorithm choice, architecture, data, hyperparameters. As P4 explained:

> *It usually happened that my currently trained model performs badly. I might roll back to the previous model, and investigate the reasons ... Data in the cloud change over time, including those we use to train models. Models might degrade due to the evolving data. We may take a look at current data and compare them with previous data to see why degradation happens.*

As a result, configuration management for ML systems becomes more complex compared to non-ML software. Besides code and dependencies, data, model files, model dependencies, hyperparameters require configuration management. The models checkpoints and data would take a large amount of space. As P8 noted, machine learning frameworks trade off exact numeric determinism for performance, "the dependent frameworks can change over time, sometimes radically". To reproduce the results, a snapshot of the whole system may be required.

4.1.6 Software Engineering Process and Management

ML and non-ML systems differ in the processes that are followed in their development. As P2 suggested, during the step of context understanding, it is important to communicate with other stakeholders about what machine learning is and is not capable of. P6 mentioned that some stakeholders might misunderstand what machine learning is capable of:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

They usually overestimate the effect of machine learning technologies. Machine learning is not a silver bullet; the effect highly depends on the data they have [✔ ✔ S21].

P14 mentioned that data processing is important to the success of the whole process [✔ S22], "garbage in garbage out, it is worth spending the time". P12 noted that data visualization plays a crucial role in understanding feature distributions and correlations as well as identifying interesting trends and patterns out of the data. As P6 noted, domain knowledge is pivotal in understanding data features:

However, sometimes domain experts are reluctant to share their knowledge. They may be afraid of being replaced by automated software or do not have any accurate reasoning but intuition.

It is hardly possible to develop a good model in a single pass (P6). The step aims to find the right balance through trial and error. Even the best machine learning practitioners need to tinker to get the models right. Sometimes, practitioners may go back to the *data* step to analyze errors.

As P3, P4, and P5 mentioned, the practitioners create an API or Web application with the machine learning model as an endpoint during production. Practitioners also need to plan for how frequently the model requires to be retrained with updated data. During the *monitor*ing step, the performance of models is tracked over time. Once data changes in a way that invalidates the models, the software should be prepared to respond to the mistakes and unexpected consequences.

Interviewees suggested that a significant difference between management of ML versus non-ML development is that the management of ML development lacks specific and practical guidance [S23]. In contrast to the development of non-ML software, development of ML systems is an iterative optimization task by nature, "there is more than one right answer" as long as the quantitative measures meet the expectation (P1). Sometimes, the available data are not sufficient to support the application context. It is impossible to achieve the expected quantitative measures no matter how good the trained model is. P6 explained:

No one knows if the quantitative metrics are achievable until we finish training our model.

Interviewees also mentioned that the development plan of ML systems is more flexible than non-ML software systems. The progress of model training is usually not in a linear way.

4.2 RQ2. Differences in Work Characteristics

In this section, we discuss differences between ML and non-ML development in terms of work characteristics. No common themes emerged from several work feature topics in our interviews or focus groups (*work scheduling autonomy*, *task variety*, *significance*, *feedback from the job*, *information processing*, *specialization*, *feedbacks from others*, *social support*, *work conditions*, *ergonomics*, *experienced meaningfulness*, *experienced responsibility*, *knowledge of results*). Thus we do not discuss them in this section.

4.2.1 Skill Variety

Interviewees identified two main differences between ML and non-ML development in terms of skill variety.

First, interviewees noted that developing machine learning frameworks and applications presented distinct technical challenges. For example, P10 suggested that, in addition to programming skills, ML development tends to require "specialized knowledge in math, information theory and statistics " [✔ S24]. P13 explained the differences as:

[For the development of non-ML software systems] developers can write code for a particular business requirement once they have learned a programming language... [For the development of ML systems] math and statistics specialized knowledge is a crucial prerequisite of designing effective models for a particular business requirement. Nevertheless, proficient programming skill is still important and could help with model implementation.

Second, interviewees suggested that a wider variety of skills is required for ML development [S25], which can make ML development more challenging if a developer lacks those skills. As P5 suggested, "in addition to programming skills, data analysis skill is required for ML development". P10 summarized that the data analysis skill consists of the abilities to acquire, clean, explore, and model data. As P6 noted, the huge volume of data in ML development brings new challenges to data analysis:

In the context of big data, performing statistical data analysis is not enough. Developers should be able to handle data analysis for a huge volume of data. For example, developers need the skills of writing distributed programs for data analysis and using distributed computing frameworks.

4.2.2 Job Complexity and Problem Solving

Interviews indicated that ML development and non-ML development present complexity in different aspects. As P12 suggested, the job complexity of non-ML development resides in architecture design and module partitioning [✔ S26]; in contrast, the job complexity of ML development resides in data modeling [S27]. P14 explained that "the architectures of distinct machine learning applications are relatively fixed, they usually consist of several modules, i.e., data collection, data preprocessing, data modeling, and testing".

The difference in job complexity leads to the difference in problem solving. Interviewees mentioned that, for non-ML development, a clear roadmap usually exists to produce a good architecture design and module partitioning [✔ S28]. Developers could then follow a step-by-step approach to implement each module (P5, P6, P8). However, for ML development, no clear roadmap exists to build effective data models (P2). As P6 suggested, the problem solving process in ML development has more uncertainties compared to non-ML development:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

We do not know what results the data can tell, to what extent a data model can be improved. We would try whatever that we think may work. Thus, the search space becomes quite large, and the workload might explode accordingly. Sometimes, more workload does result in better performance.

4.2.3 Task Identity

Interviewees reported few differences in terms of task identity. One difference is suggested by P4 and P5, who reported that it is harder to make an accurate plan for tasks in ML development [✔ S29]. P4 summarized the reasons as:

In non-ML software development, the project can be divided into distinct tasks according to function points. Developers could easily tell how long it will take to finish the implementation of a particular function point. However, in machine learning development, data modeling is an indivisible task. To achieve acceptable performance, the search space is usually quite large. Making an accurate plan for such a task is hard.

Besides, interviewees noted that ML developers have less control over their task progress towards target performance (P9, P10) [S30]. Once starting data modeling in ML development, hard work may not always consistently lead to satisfying results (P4).

4.2.4 Interaction Outside the Organization

Interviewees reported that ML developers face more challenges when communicating with customers [✔ S31]. As P2 explained:

It is harder to communicate the project progress for machine learning development due to the non-linear process of data modeling... The results of machine learning development are not straightforward to interpret. For example, it is difficult to explain why a neural network works for image processing.

4.3 Survey Results

We summarize the survey results in Table 2. The *Statement* column shows the statements presented to respondents. The following column indicates the labels we used to identify statements throughout the paper. The four *Likert Distribution* subcolumns present the distribution of agreement for each group of respondents (ML practitioners - *ML*, Non-ML Practitioners - *Non-ML*, practitioners of ML Framework/Tool/Library - *ML FTL*, and Practitioners of ML Application - *ML App*). For the Likert distributions, the leftmost bar indicates strong disagreement, the middle bar indicates neutrality, and the rightmost bar indicates the strongest agreement. For example, most machine learning practitioners strongly agree with S24.

The *P-value* column indicates whether the differences in the agreement for each statement are statistically significant between ML and non-ML in the first subcolumn, and ML FTL and ML App in the second subcolumn. The table is sorted by the p-values with Benjamini-Hochberg correction in the first subcolumn. Statistically significant differences at a 95% confidence level (Benjamini-Hochberg corrected p-value < 0.05) are highlighted in green.

TABLE 3: Interpretation of Cliff's delta value.

Cliff's Delta Value	Interpretation
$\|\delta\| < 0.147$	Negligible
$0.147 \leq \|\delta\| < 0.330$	Small
$0.330 \leq \|\delta\| < 0.474$	Medium
$\|\delta\| \geq 0.474$	Large

The *Effect Size* column indicates the difference between ML and non-ML in the first sub-column, and the difference between ML FTL and ML App in the second subcolumn. We use Cliff's delta to measure the magnitude of the differences since Cliff's delta is reported to be more robust and reliable than Cohen's delta [32]. Cliff's delta represents the degree of overlap between two sample distributions, ranging from -1 to $+1$. The extreme value ± 1 occurs when the intersection between both groups is an empty set. When the compared groups tend to overlap, Cliff's Delta approaches zero. Effect sizes are additionally colored on a gradient from blue to orange based on the magnitudes of difference as referring to the interpretation of Cliff's delta in Table 3: blue color means the former group is more likely to agree with the statement, and orange color means the latter group is more likely to agree with the statement.

Overall, the results of the survey confirm some differences in interviewees' claims. Based on the observed statistically significant differences between ML and non-ML development, we can say with some certainty that:

- **Requirements**: Collecting requirements in the development of ML systems involves more preliminary experiments, and creates a need for the predictable degradation in the performance. [✔ S2, ✔ S3]
- **Design**: Detailed design of ML systems is more time-consuming and tends to be conducted in an intensively iterative way. [✔ S7]
- **Testing and Quality**: Collecting a testing dataset requires more effort for ML development; Good performance during testing cannot guarantee the performance of ML systems in production. [✔ ✔ S15, ✔ S17]
- **Process and Management**: The availability of data usually limits the capability of ML systems; Data processing tends to be more important to the success of the whole process. [✔ ✔ S21, ✔ S22]
- **Skill Variety**: ML development intensively requires knowledge in math, information theory, and statistics. [✔ S24]
- **Job Complexity and Problem Solving**: ML practitioners have a less clear roadmap for building systems. [✔ S28]
- **Task Identity**: It is much harder to make an accurate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

TABLE 2: Survey Results. Orange cells indicate where the former group (ML practitioners/ML FTL practitioners) disagrees more strongly with the statement than the latter group (non-ML practitioners/ML App practitioners); blue cells indicate where the former group agrees more strongly. Green cells represent statistically significant differences. The number in "()" indicates the size of each group.

Statement		Likert Distributions ML (98)	Non-ML (244)	ML FTL (39)	ML App (59)	Cliff's Delta ML vs. Non-ML	ML FTL vs. ML App	P-values ML vs. Non-ML	ML FTL vs. ML App
Developing my software requires knowledge in math, information theory and statistics.	S24					0.45	-0.19	✔ .000	.320
Detailed design is time-consuming and conducted in an iterative way.	S7					0.32	0.18	✔ .000	.271
Requirements should consider predictable degradation in the performance of software.	S3					0.29	0.11	✔ .000	.433
It is easy to make an accurate plan for the development tasks of my software.	S29					-0.32	0.03	✔ .000	.779
Data processing is important to the success of the whole development process.	S22					0.26	-0.20	✔ .000	.271
Collecting testing dataset is labor intensive.	S15					0.27	-0.26	✔ .000	.188
Developing my software requires frequent communications with the clients.	S31					-0.29	-0.14	✔ .000	.577
My software is tested by using automated testing tools.	S18					0.26	0.48	✘ .000	✔ .001
Good testing results can guarantee the performance of my software in production.	S17					-0.23	0.09	✔ .001	.482
Available data limit the capability my software.	S21					0.22	-0.48	✔ .001	✔ .001
Collecting requirements involve a large number of preliminary experiments.	S2					0.20	0.09	✔ .002	.577
A clear roadmap exists to build my software.	S28					-0.24	0.07	✔ .002	.661
High level architectural design is relatively fixed.	S4					-0.20	0.07	✘ .017	.719
Creativity is important during debugging.	S11					0.12	0.07	.064	.719
My team puts a lot of effort into maintenance of my software.	S19					-0.15	0.21	.065	.188
The higher the performance measures are, the better my software is.	S16					0.08	0.19	.068	.271
Architecture design is complicated for my software.	S26					0.10	0.32	.069	✔ .047
Data modeling is complicated for my software.	S27					0.07	-0.15	.077	.471
Detailed design is flexible.	S6					0.08	0.11	.107	.459
Requirements of my software are uncertain.	S1					0.10	0.11	.151	.482
Testing involves multiple runs of my software to gather a population of quantitative measures.	S14					0.07	0.06	.152	.719
My coding workload is heavy.	S8					0.07	0.08	.223	.665
I have control over the progress towards the target performance.	S30					0.06	0.02	.265	.756
Code reuse happens frequently across different projects.	S9					0.06	-0.12	.265	.482
Debugging aims to locate and fix bugs in my software.	S10					0.06	-0.01	.358	.943
Creating my software requires a team of people, each with different skills.	S25					0.04	0.09	.540	.482
Testing results of my software are hard to reproduce.	S13					0.04	0.04	.546	.787
Low coupling in the components of my software is important.	S5					-0.01	0.08	.861	.459
Configuration management are mainly for the code.	S20					-0.07	-0.14	.894	.943
Software engineering management lacks practical guidance for my software.	S23					-0.01	-0.20	.894	.482
Bugs in my software usually reside in the code.	S12					-0.01	-0.05	.979	.787

plan for the tasks for ML development. [✔ S29]

- **Interaction**: ML practitioners tend to communicate less frequently with clients. [✔ S31]

The survey results cannot confirm several of interviewees' claims about differences between ML and non-ML. For example, requirement is deterministic across ML system development and non-ML software development [S1]. One explanation is that although there is uncertainty in aspects of the models that ML developers ship, requirements themselves are deterministic.

5 Discussion

5.1 Implications

Embracing Uncertainty. As mentioned by our interviewees, uncertainty lies in various aspects of the development of ML systems.

First, uncertainty comes from the data as part of the requirement. Although a development team of ML system has a target to attain, e.g., building a speech recognition software with absolute precision, a bunch of preliminary experiments is required to make sure the goal is achievable, and the available data suffice for the target. Understanding application contexts, quick data visualization and hands-on experimental data modeling on a small-scale dataset could be helpful to accelerate the progress of preliminary experiments during requirement gathering and analysis phase. Instead of trying a number of tools, it might be wiser for machine learning practitioners to focus on a few tools to learn and use [22]. The exploratory process of preliminary experiments in ML development is similar to scientific programming [9] and may benefit from the lessons learned from scientific programming.

Second, uncertainty originates in the inherent randomness of machine learning algorithms. Machine learning practitioners should shift their mindset and embrace uncertainty. For instance, a machine learning algorithm may be initialized to a random state; random noise helps to effectively find optimized solution during gradient descent (stochastic gradient descent). To reduce uncertainty, machine learning practitioners could achieve reproducibility to some extent by using the same code, data, and initial random state. Thus, version control toolchains for code, data and parameters are essential to achieve reproducibility [5]. However, storing all states may introduce significant overhead and slow down the development process. Thus, the effectiveness of such toolchains is subject to future investigation. In addition, to evaluate the performance of a machine learning algorithm, practitioners usually randomly split the data into a training and test set or use k-fold cross-validation. The performance of a machine learning algorithm should be reported as a distribution of measures, rather than a single value, as emphasized by our participants.

Handling Data. As discussed by our interviewees, data play a vital role in the development of ML systems. The large quantity of training data can have a tremendous impact on the performance of ML systems. As confirmed by our participants, data collection becomes one of the critical challenges for the development of ML systems. Data collection literature focuses on three lines of research [31]: data acquisition techniques to discover, augment, or generate datasets, data labeling techniques to label individual data points, and transfer learning techniques to improve existing datasets. Future studies could integrate existing data collection techniques into the process of ML development. In addition, existing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

data collection techniques tend to be application or data specific. More effort is needed to generalize those proposed techniques to various applications. ML practitioners also used distributed platforms to process a large quantity of data in parallel. Debugging the parallel computations for data modeling is time-consuming and error-prone. As illustrated in [16], future studies could put more effort to facilitate interactive and real-time debugging for ML systems.

Along with data quantity, quality is also critical to build a powerful and robust ML system. "Garbage in, garbage out", what practitioners obtain from the machine learning software is a representation of what they feed into the software. Real-world data is comprising of missing values, imbalanced data, outliers, etc. It becomes imperative that machine learning practitioners process the data before building models. Future research could develop data visualization tools that give an overview of the data, help in locating irregularities, enable practitioners to focus on where the data actually needs cleansing. However, high-quality datasets during development cannot ensure the high performance of machine learning systems eternally. Within a rapidly evolving environment, a machine learning system degrades in the accuracy as soon as the software is put in production [34]. Practitioners need to recognize that there is never a final version of a machine learning system, which needs to be updated and improved continuously over time (e.g., feeding new data and retrain models). Online feedback and performance measurement of ML systems are fertile areas for future research.

5.2 Threats to Validity

Internal Validity. It is possible that some of our survey respondents had a poor understanding of the statements for rating. Their responses may introduce noise to the data that we collected. To reduce the impact of this issue, we included an "I don't know" option in the survey and ignored responses marked as such. We also dropped responses that were submitted by people whose job roles are none of these: software development, testing and project management. Two of the authors translated our survey to Chinese to ensure that respondents from China could understand our survey well. To reduce the bias of presenting survey bilingually, we carefully translated our survey to make sure there is no ambiguity between English and Chinese terms. We also polished the translation by improving clarity and understandability according to the feedback from our pilot survey.

The effect sizes of statistically significant differences between ML and non-ML development reported in this work range from negligible to medium. The negligible effect size indicates that a particular difference between machine learning and non-machine learning development is trivial, even if it is statistically significant. To mitigate this threat, we did not emphasize those differences in our results.

As we selected survey respondents, we sent invitations to a variety of potential respondents that might involve in different parts of ML ecosystems (ML frameworks, tools, and libraries, software applications with ML components). We mixed the responses from those ML respondents when we studied the differences between ML and non-ML development. It is possible that differences exist in the perceptions of these two groups, and overwhelm ML versus non-ML differences. To prevent this threat, we compared the differences in distributions of Likert responses between these two groups.

To recruit respondents from both ML and non-ML populations, we spread the survey broadly to a wide range of companies from various locations around the world. In the beginning of the survey, we articulated that the purpose of our study is to understand whether and how machine learning changes software engineering practices. This description may attract more attention from a part of the non-ML population, who know about ML, but ML is not part of their daily work. In addition, the description may generate a tacit presumption that machine learning changes software engineering practices. The presumption may mislead the respondents to exaggerate the differences they perceived.

External Validity. To improve the generalizability of our findings, we interviewed 14 interviewees from three companies, and surveyed 342 respondents from 26 countries across four continents who are working for various companies (e.g., Amazon, Alibaba, Baidu, Google, Hengtian, IBM, Intel, IGS, Kodak, Lenovo, Microsoft, and Morgan Stanley) or contributing to open source machine learning projects that are hosted on GitHub, in various roles.

We wish though to highlight that while we selected employees from three Chinese IT companies for our interviews, we improved the responses from interviews through focus group discussions that involved more IT companies, and the surveyed population is considerably wide. The improved responses from the interviews were used to bootstrap the statements to rate in our survey. The survey permitted respondents to add additional comments whenever appropriate via free-form fields; looking at the responses in such fields we do not observe any signs of missing statements.

In addition, some reported claims from our interviewees were not validated through the survey and might be premature.

6 Related Work

Some prior work provides prescriptive practices for the development of machine learning systems (e.g., [29]). Some discuss the realistic challenges and best practices in the industry, e.g., machine learning model management at Amazon [33], and best practices of machine learning engineering at Google [40]. Some investigated machine learning related questions on Stack Overflow [38]. These works are based on the experience of the authors and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2019.2937083, IEEE Transactions on Software Engineering

largely do not contextualize machine learning development as a special type of software engineering. In contrast, our findings are based on empirical observations that explicitly focus on the differences between ML and non-ML development.

Like our work, several researchers have conducted empirical studies of software engineering for data science. Some focus on how data scientists work inside a company via interviews to identify pain points from a general tooling perspective [13], and explore challenges and barriers for adopting visual analytic tools [20]. Other focus on characterizing professional roles and practices regarding data science: Harris et al. [17] surveyed more than 250 data science practitioners to categorize data science practitioners and identify their skill sets. Kim et al. interviewed sixteen data scientists at Microsoft to identify five working styles [21], and supplement Harris et al.'s survey with tool usage, challenges, best practices and time spent on different activities [22]. In contrast to this prior work, our paper studies broad differences between ML and non-ML development.

Most similar to our study is an empirical investigation of integrating AI capabilities into software and services and best practices from Microsoft teams [4]. From their proposed ML workflow, they identified 11 challenges, including 1) data availability, collection, cleaning, and management, 2) education and training, 3) hardware resources, 4) end-to-end pipeline support, 5) collaboration and working culture, 6) specification, 7) integrating AI into larger systems, 8) guidance and mentoring, 9) AI tools, 10) scale, and 11) model evolution, evaluation, and deployment. The identified challenges emerge across different software development practices. Our findings differ from theirs in a number of areas:

- **Design.** Both studies agree that maintaining modularization in ML development is difficult. Their study [4] summarized the reasons as the low extensibility of an ML model and the non-obvious interaction between ML models. In contrast, we found ML development places comparable emphasis on low coupling in components as non-ML development [S5]. Both studies agree that the rapid iterations exist in the detailed design of ML systems [✔ S7].
- **Construction and Tools.** Both studies agree that code reuse in ML development is challenging due to varying application context and input data. Despite the challenge of code reuse in ML development, we found that code reuse happens in ML development as frequently as in non-ML development [S9].
- **Process and Management.** Both studies agree that management of ML development is challenging due to the involvement of data, and that the availability of data usually limits the capability of ML systems [✔ ✔ S21].
- **Configuration Management.** Both studies agree that data versioning is required in ML development. Despite the necessity of data versioning, we found that current configuration management activities in ML development still focuses on code versioning [S20]. In addition to data versioning, the earlier study [4] suggested to keep track of how data is gathered and processed.

As discussed above, our study confirms some of the findings reported in Amershi et al.'s work. Different from Amershi et al.'s work, since we followed the SWEBOK and considered work characteristics from applied psychology domain in our interviews, we recognized the differences between ML and non-ML development in other aspects, e.g., requirement gathering, job complexity, problem solving process, and task identity. Moreover, we collected perceptions from broader population groups, e.g., involving open source practitioners and professionals from various software industries.

7 Conclusion

In this work, we identified the significant differences between ML and non-ML development. The differences lie in a variety of aspects including software engineering practices (e.g., exploratory requirements elicitation and iterative processes in ML development) and the context of software development (e.g., high complexity and demand for unique solutions and ideas in ML development). The differences originate from inherent features of machine learning - uncertainty and the data for use.

To tackle uncertainty, ML practitioners should shift their mindset, and embrace the uncertainty in preliminary experiments and the randomness of ML algorithms. They could learn the lessons from scientific programming, which also involves exploratory processes in the development. In addition, version control toolchains for code, data and parameters could play a vital role for ML practitioners to achieve reproducibility. ML practitioners should also devote sufficient effort to handle the data for use. Future studies could put more effort to provide interactive and real-time debugging tools to facilitate efficient development of ML systems. To deal with the rapid evolution of data, online feedback and performance measurement for ML systems are fertile areas for future research.

In a larger sense, this work represents a step towards understanding software development not as a homogenous bulk, but as a rich tapestry of varying practices that involve people of diverse backgrounds across various domains. Precise differences may reside in different kinds of ML architectures. We leave these questions to future studies.

Acknowledgment

The authors would like to thank all interviewees for their participation and survey participants for responding to our survey.

REFERENCES

[1] Nvivo qualitative data analysis software, 2018.

[2] The amazon machine learning process. https://docs.aws.amazon.com/machine-learning/latest/dg/the-machine-learning-process.html, 2019. Mar. 2019.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[4] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *Proceedings of the 39th International Conference on Software Engineering - SEIP track*. IEEE Computer Society, May 2019.

[5] A. Anjos, M. Günther, T. de Freitas Pereira, P. Korshunov, A. Mohammadi, and S. Marcel. Continuously reproducing toolchains in pattern recognition and machine learning experiments. In *Proceedings of the 34th International Conference on Machine Learning*, Aug. 2017. https://openreview.net/group?id=ICML.cc/2017/RML.

[6] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395. ACM, 2017.

[7] P. Bourque, R. E. Fairley, et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[8] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[9] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th International Conference on Software Engineering*, pages 550–559. IEEE Computer Society, 2007.

[10] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[11] G. Ericson. What is the team data science process? https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview, 2017. Mar. 2019.

[12] A. Ferlitsch. Making the machine: the machine learning lifecycle. https://cloud.google.com/blog/products/ai-machine-learning/making-the-machine-the-machine-learning-lifecycle, 2019. Mar. 2019.

[13] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics. *interactions*, 19(3):50–59, May 2012.

[14] Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[15] G. Guest, A. Bunce, and L. Johnson. How many interviews are enough? an experiment with data saturation and variability. *Field methods*, 18(1):59–82, 2006.

[16] M. A. Gulzar, M. Interlandi, S. Yoo, S. D. Tetali, T. Condie, T. Millstein, and M. Kim. Bigdebug: Debugging primitives for interactive big data processing in spark. In *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering*, pages 784–795. IEEE, 2016.

[17] H. Harris, S. Murphy, and M. Vaisman. *Analyzing the Analyzers: An Introspective Survey of Data Scientists and Their Work*. " O'Reilly Media, Inc.", 2013.

[18] S. E. Humphrey, J. D. Nahrgang, and F. P. Morgeson. Integrating motivational, social, and contextual work design features: a meta-analytic summary and theoretical extension of the work design literature. *Journal of applied psychology*, 92(5):1332, 2007.

[19] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

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

[21] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, pages 96–107. ACM, 2016.

[22] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, 44(11):1024–1038, 2018.

[23] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer, 2008.

[24] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[25] K.-c. Lee, B. Orten, A. Dasdan, and W. Li. Estimating conversion rate in display advertising from past performance data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 768–776. ACM, 2012.

[26] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 120–131. ACM, 2018.

[27] S. Ma, Y. Aafer, Z. Xu, W.-C. Lee, J. Zhai, Y. Liu, and X. Zhang. Lamp: data provenance for graph based machine learning algorithms through derivative computation. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 786–797. ACM, 2017.

[28] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama. Mode: automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 175–186. ACM, 2018.

[29] A. Ng. *Machine learning yearning*. URL: http://www. mlyearning. org, 2017.

[30] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[31] Y. Roh, G. Heo, and S. E. Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *arXiv preprint arXiv:1811.03402*, 2018.

[32] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohensd for evaluating group differences on the nsse and other surveys. In *Proceedings of the Annual Meeting of the Florida Association of Institutional Research*, pages 1–33, 2006.

[33] S. Schelter, F. Biessmann, T. Januschowski, D. Salinas, S. Seufert, G. Szarvas, et al. On challenges in machine learning model management. http://sites.computer.org/debull/A18dec/p5.pdf, 2018. Mar. 2019.

[34] D. Talby. Lessons learned turning machine learning models into real products and services. https://www.oreilly.com/ideas/lessons-learned-turning-machine-learning-models-into-real-products-and-services, 2018. Mar. 2019.

[35] R. Thomas. What do machine learning practitioners actually do? https://www.fast.ai/2018/07/12/auto-ml-1/, 2018. Mar. 2019.

[36] Y. Tian, K. Pei, S. Jana, and B. Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. ACM, 2018.

[37] P. K. Tyagi. The effects of appeals, anonymity, and feedback on mail survey response patterns from salespeople. *Journal of the Academy of Marketing Science*, 17(3):235–241, Jun 1989.

[38] Z. Wan, J. Tao, J. Liang, Z. Cai, C. Chang, L. Qiao, and Q. Zhou. Large-scale empirical study on machine learning related questions on stack overflow. *Journal of ZheJiang University (Engineering Science)*, 53(5):819–828, 2019.

[39] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4):544–558, 2011.

[40] M. Zinkevich. Rules of machine learning: Best practices for ml engineering. https://developers.google.com/machine-learning/guides/rules-of-ml/, 2018. Mar. 2019.