"We have no idea how models will behave in production until production": How engineers operationalize machine learning

SHREYA SHANKAR*, University of California, Berkeley, USA
ROLANDO GARCIA*, University of California, Berkeley, USA
JOSEPH M. HELLERSTEIN, University of California, Berkeley, USA
ADITYA G. PARAMESWARAN, University of California, Berkeley, USA

Organizations rely on machine learning engineers (MLEs) to deploy models and maintain ML pipelines in production. Due to models' extensive reliance on fresh data, the operationalization of machine learning, or MLOps, requires MLEs to have proficiency in data science and engineering. When considered holistically, the job seems staggering—how do MLEs do MLOps, and what are their unaddressed challenges? To address these questions, we conducted semi-structured ethnographic interviews with 18 MLEs working on various applications, including chatbots, autonomous vehicles, and finance. We find that MLEs engage in a workflow of (i) data preparation, (ii) experimentation, (iii) evaluation throughout a multi-staged deployment, and (iv) continual monitoring and response. Throughout this workflow, MLEs collaborate extensively with data scientists, product stakeholders, and one another, supplementing routine verbal exchanges with communication tools ranging from Slack to organization-wide ticketing and reporting systems. We introduce the 3Vs of MLOps: *velocity*, *visibility*, and *versioning*—three virtues of successful ML deployments that MLEs learn to balance and grow as they mature. Finally, we discuss design implications and opportunities for future work.

CCS Concepts: • **Human-centered computing** → **Human computer interaction (HCI)**; User studies.

Additional Key Words and Phrases: mlops, interview study

ACM Reference Format:
Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2023. *"We have no idea how models will behave in production until production"*: How engineers operationalize machine learning. *Proc. ACM Hum.-Comput. Interact.* 37, 4, Article 111 (August 2023), 34 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As machine learning (ML) models are increasingly incorporated into software, a nascent sub-field called *MLOps* (short for ML Operations) has emerged to organize the "set of practices that aim to deploy and maintain ML models in production reliably and efficiently" [2, 104]. It is widely recognized that MLOps issues pose challenges to organizations. Anecdotal reports claim that 90% of ML models don't make it to production [103]; others claim that 85% of ML projects fail to deliver value [94]—signaling the fact that translating ML models to production is difficult.

*Both authors contributed equally to this research.

Authors' addresses: Shreya Shankar, University of California, Berkeley, Berkeley, CA, USA, shreyashankar@berkeley.edu; Rolando Garcia, University of California, Berkeley, Berkeley, CA, USA, rogarcia@berkeley.edu; Joseph M. Hellerstein, University of California, Berkeley, Berkeley, CA, USA, hellerstein@berkeley.edu; Aditya G. Parameswaran, University of California, Berkeley, Berkeley, CA, USA, adityagp@berkeley.edu.

2573-0142/2023/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

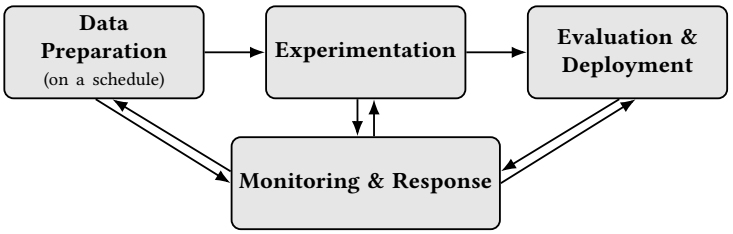


Fig. 1. Core tasks in the MLOps workflow. Prior work discusses a production data science workflow of preparation, modeling, and deployment [102]. Our work exposes (i) the scheduled and recurring nature of **data preparation** (including automated ML tasks, such as model retraining), identifies (ii) a broader **experimentation** step (which could include modeling or adding new features), and provides more insight into human-centered (iii) **evaluation & deployment**, and (iv) **monitoring & response**.

At the same time, it is unclear *why* MLOps issues are difficult to deal with. Our present-day understanding of MLOps is limited to a fragmented landscape of white papers, anecdotes, and thought pieces [18, 21, 24, 61], as well as a cottage industry of startups aiming to address MLOps issues [34]. Early work by Sculley et al. [87] attributes MLOps challenges to *technical debt*, analogous to that in software engineering but exacerbated in ML. Prior work has studied general practices of data scientists working on ML [37, 66, 85, 110], but successful ML deployments seem to further involve a "team of engineers who spend a significant portion of their time on the less glamorous aspects of ML like maintaining and monitoring ML pipelines" —that is, ML engineers (MLEs) [75]. It is well-known that MLEs typically need to have strong data science and engineering skills [3], but it is unclear how those skills are used in their day-to-day workflows.

There is thus a pressing need to bring clarity to MLOps—specifically in identifying what MLOps typically involves—across organizations and ML applications. While papers on MLOps have described specific case studies, prescribed best practices, and surveyed tools to help automate the ML lifecycle, there is a pressing need to understand the *human-centered* workflow required to support and sustain the deployment of ML models in practice. A richer understanding of common practices and challenges in MLOps can surface gaps in present-day processes and better inform the development of next-generation ML engineering tools. To address this need, we conducted a semi-structured interview study of ML engineers (MLEs), each of whom has been responsible for a production ML model. With the intent of identifying common themes across organizations and industries, we sourced 18 ML engineers from different companies and applications, and asked them open-ended questions to understand their workflow and day-to-day challenges—both on an individual and organizational level.

Prior work focusing on the earlier stages of data science has shown that it is a largely iterative and manual process, requiring humans to perform several stages of data cleaning, exploration, model building, and visualization [30, 46, 73, 85]. Before embarking on our study, we expected that the subsequent deployment of ML models in production would instead be more amenable to automation, with less need for human intervention and supervision. Our interviews, in fact, revealed the opposite—much like the earlier stages of data science, deploying and maintaining models in production is highly iterative, manually-intensive, and team-oriented. Our interviewees emphasized organizational and collaborative strategies to sustain ML pipeline performance and minimize pipeline downtime, mentioning on-call rotations, manual rules and guardrails, and teams of practitioners inspecting data quality alerts.

In this paper, we provide insight into human-centered aspects of MLOps practices and identify opportunities for future MLOps tools. We conduct a semi-structured interview study solely focused on ML engineers, an increasingly important persona in the broader software development ecosystem as more applications leverage ML. Our focus on MLEs, and uncovering their workflows and challenges as part of the MLOps process, addresses a gap in the literature. Through our interviews, we characterize an ML engineer's typical workflow (on top of automated processes) into four stages (Figure 1): (i) data preparation, (ii) experimentation, (iii) evaluation and deployment, and (iv) monitoring and response, all centered around team-based, collaborative practices. Key takeaways for each stage are as follows:

Data ingestion often runs automatically, but MLEs drive data preparation through data selection, analysis, labeling, and validation (Section 4.1). We find that organizations typically leverage teams of data engineers to manage recurring end-to-end executions of data pipelines, allowing MLEs to focus on ML-specific steps such as defining features, a retraining cadence, and a labeling cadence. If a problem can be automated away, engineers prefer to do so—e.g., retraining models on a regular cadence to protect against changes in the distribution of features over time. Thus, they can spend energy on tasks that require human input, such as supervising crowd workers who provide input labels or resolve inconsistencies in these labels.

Even in production, experimentation is highly iterative and collaborative, despite the use of model training tools and infrastructure (Section 4.2). As mentioned earlier, various articles claim that it is a problem for 90% of models to never make it to production [103], but we find that this statistic is misguided. The nature of constant experimentation is bound to create many versions of models, a small fraction of which (i.e. "the best of the best") will make it to production. MLEs discussed exercising judgment when choosing next experiments to run, and expressed reservations about AutoML tools, or "keeping GPUs warm," given the vast search space. MLEs consult domain experts and stakeholders in group meetings, and prefer to iterate on the data (e.g., to identify new feature ideas) over innovating on model architectures.

Organizations employ a multi-stage model evaluation and deployment process, so MLEs manually review and authorize deployment to subsequent stages (Section 4.3). Textbook model evaluation "best practices" do not do justice to the rigor with which organizations think about deployments: they generally focus on using one typically-static held-out dataset in an offline manner to evaluate the model on [54] and a single ML metric choice (e.g., precision, recall) [68]. We find that many MLEs carefully deploy changes to increasing fractions of the population in stages. At each stage, MLEs seek feedback from other stakeholders (e.g., product managers and domain experts) and invest significant resources in maintaining multiple up-to-date evaluation datasets and metrics over time—especially to ensure that data sub-populations of interest are adequately covered.

MLEs closely monitor deployed models and stand by, ready to respond to failures in production (Section 4.4). MLEs ensured that deployments were reliable via strategies such as on-call rotations, data monitoring, and elaborate rule-based guardrails to avoid incorrect outputs. MLEs discussed pain points such as alert fatigue from alerting systems and the headache of managing pipeline jungles [87], or amalgamations of various filters, checks, and data transformations added to ML pipelines over time.

The rest of our paper is organized as follows: we cover background and work related to MLOps from the CSCW, HCI, ML, software engineering, and data science communities (Section 2). Next, we describe the methods used in our interview study (Section 3). Then, we present our results and discuss our findings, including opportunities for new tooling (Section 4 and Section 5). Finally, we conclude with possible areas for future work.

2 RELATED WORK

Our work builds on previous studies of data and ML practitioners in industry. We begin with the goal of characterizing the role of an ML Engineer, starting with related data science roles in the literature and drawing distinctions that make MLEs unique. We then review work that discusses data science and ML workflows, not specific to MLOps. Third, we cover challenges that arise from productionizing ML. Fourth, we survey software engineering practices in the literature that tackle such challenges. Finally, we review recent work that explicitly attempts to define and discuss MLOps practices.

2.1 Characterizing the ML Engineer

Data science roles span various engineering and research tasks [40], and many data-related activities are performed by people without "data" or "ML" in their job title [41], so it can be hard to clearly define job descriptions [66]. Nonetheless, since we focus on production ML pipelines, we discuss personas related to data science, ML, and engineering—culminating in the description of the persona we study.

The Data Scientist: Multiple studies have identified subtypes of data scientists, some of whom are more engineering-focused than others [40, 110]. Zhang et al. [110] describe the many roles data scientists can take—communicator, manager/executive, domain expert, researcher/scientist, and engineer/analyst/programmer. They found considerable overlap in skills and tasks performed between the (i) engineer/analyst/programmer and (ii) researcher/scientist roles: both are highly technical and collaborate extensively. Separately, Kim et al. taxonomized data scientists as: insight providers, modeling specialists, platform builders, polymaths, and team leaders [40]. Modeling specialists build predictive models using ML, and platform builders balance both engineering and science as they produce reusable software across products.

The Data Engineer: While data scientists engage in activities like exploratory data analysis (EDA), data wrangling, and insight generation [37, 106], *data engineers* are responsible for building robust pipelines that regularly transform and prepare data [51]. Data engineers often have a software engineering and data systems background [40]. In contrast, data scientists typically have modeling, storytelling, and mathematics backgrounds [40, 51]. Since production ML systems involve data pipelines and ML models in larger software services, they require a combination of data engineering, data science, and software engineering skills.

The ML Engineer (MLE): Our interview study focuses on the distinct *ML Engineer* (MLE) persona. MLEs have a multifaceted skill set: they know how to transform data as inputs to ML pipelines, train ML models, serve models, and wrap these pipelines in software repositories [35, 44, 81]. MLEs need to regularly process data at scale (much like data engineers [44]), employing statistics and ML techniques as do data scientists [74], and are responsible for production artifacts as are software engineers [52]. Unlike typical data scientists and data engineers, MLEs are responsible for deploying ML models and maintaining them in production.

We classify production ML into two modes. One, which we call *single-use* ML, is more client-oriented, where the focus is to generate predictions once to make a specific data-informed business decision [46]. Typically, this involves producing reports, primarily performed by data scientists [32, 41]. In the other mode, which we call *repeated-use* ML, predictions are repeatedly generated, often as intermediate steps in data pipelines or as part of ML-powered products, such as voice assistants and recommender systems [3, 40]. Continuously generating ML predictions over time requires

more data and software engineering expertise [41, 99]. In our study, we focus on MLEs who work on the latter mode of production ML.

2.2 Machine Learning Workflows

Here, we cover literature on ML practitioners' workflows. We discuss both technical and collaborative workflows, and then we describe the workflow our study seeks to uncover.

Several studies have investigated aspects of the broader ML workflow, mostly in single-use production ML applications. Early studies on data science workflows point to industry-originated software development process models, such as the Agile framework [13] and the CRoss Industry Standard Process for Data Mining (CRISP-DM) [10]. More recently, Studer et al. [96] introduce CRISP-ML, a new process model that augments CRISP-DM with a final "monitoring and maintenance" phase to support ML workflows. Muller et al. [66] interview practitioners and focus on the data practices of data science workflows, breaking them down into discovery, capture, design, and curation. Wongsuphasawat et al. [106]'s workflow includes some ML: it consists of data acquisition, wrangling, exploration, modeling, and reporting. Wang et al. [102] takes another step back and includes productionization; they identify three high-level phases of preparation, modeling, and deployment. Preparation includes activities ranging from wrangling [36] to feature engineering [74]. Modeling includes selection, hyperparameter optimization, ensembling, and validation, and deployment includes monitoring and improvement [102, 110]. Of the three large stages, several studies have identified preparation as the most time-intensive stage of the workflow [27, 66], where data scientists commonly iterate on rules to help generate features [30, 72].

While the above stages of the data science workflow comprise a loop of technical tasks, Kross and Guo [46] identify an *outer loop* of data science, centered around collaborative practices. The outer loop consists of groundwork (i.e., building trust), orienting, problem framing, magic (i.e., technical loop), and counseling. While Kross and Guo [46]'s loop focuses on data science work that directly interacts with clients, mostly in the form of single-use ML, similar themes emerge when performing repeated-use ML engineering work, e.g., repeatedly generating ML predictions in an automated fashion. In production settings, predictions must yield value for the business [74], requiring some groundwork, orienting, and problem framing. In their paper on tensions around collaborative, applied data science work, Passi and Jackson [73] discuss that it's important to align different stakeholders on system performance metrics: for example, one of their interviewees mentioned that accuracy is a "problematic" metric because different users interpret it differently. In another example, Holstein et al. [31] say that a single global metric doesn't capture performance for certain groups of users (e.g., accuracy for a subgroup might decrease when overall accuracy increases).

In our study, we characterize the workflow from a repeated-use ML engineering perspective, focusing on specific practices within deployment stages. Some related work defines steps in the ML workflow, such as model training and model monitoring, through both short papers [59] and extensive literature reviews [48]. We take a different but complementary approach: like Muller et al. [66] who focus on data scientists, we conduct an interview study of MLEs, using grounded theory to analyze our findings [95]. Further, our study seeks to uncover collaborative practices and challenges, focusing on the ML engineering perspective, and how MLEs align all stakeholders such that ML systems continually generate value.

2.3 Production ML Challenges

Sculley et al. [87] were early proponents that production ML systems raise special challenges and can be hard to maintain over time, based on their experience at Google. They coined the "Changing Anything Changes Everything" (CACE) principle: if one makes a seemingly innocuous change to an ML system, such as changing the default value for a feature from 0 to -1, the entire system's

behavior can drastically change. CACE easily creates technical debt and is often exacerbated as errors "cascade," or compound, throughout an end-to-end pipeline [71, 75, 76, 85, 87]. We cover three well-studied challenges in production ML: data quality, reproducibility, and specificity.

First, ML predictions are only as good as their input data [75, 76], requiring active efforts from practitioners to ensure good data quality [8]. Xin et al. [107] observe that production ML pipelines consist of models that are automatically retrained, and we find that this retraining procedure is a pain point for practitioners because it requires constant monitoring of data. If a model is retrained on bad data, all future predictions will be unreliable. Data distribution shift is another known data-related challenge for production ML systems [63, 69, 78, 97, 105], and our work builds on top of the literature by reporting on how practitioners tackle shift issues.

Next, reproducibility in data science workflows is a well-understood challenge, with attempts to partially address it [9, 16, 33, 43]. Recent work also indicates that reproducibility is an ongoing issue in data science and ML pipelines [4, 39, 47, 84]. Kross and Guo [47] mention that data science educators who come from industry specifically want students to learn how to write "robust and reproducible scientific code." In interview studies, Xin et al. [108] observe the importance of reproducibility in AutoML workflows, and Sambasivan et al. [85] mention that practitioners who create reproducible data assets avoid some errors.

Finally, other related work has identified that production ML challenges can be specific to the ML application at hand. For example, Sambasivan et al. [85] discusses how, in high-stakes domains like autonomous vehicles, data quality is extra important and explicitly requires collaboration with domain experts. They explain how data errors compound and have disastrous impacts, especially in resource-constrained settings. Unlike the present study, their focus is on data quality issues as opposed to understanding typical MLE workflows and challenges. Paleyes et al. [71] review published reports of individual ML deployments and mention that not all ML applications can be easily tested prior to deployment. While ad recommender systems might be easily tested online with a small fraction of users, other applications require significant simulation testing depending on safety, security, and scale issues [49, 70]. Common applications of ML, such as medicine [77], customer service [19], and interview processing [6] , have their own studies. Our work expands on the literature by identifying common challenges across various applications and reporting on how MLEs handle them.

2.4 Software Engineering for ML

Through interviews and practitioner surveys, some papers explore, at a high level, how ML engineering practices differ from traditional software engineering practices. Hill et al. [29] interview ML application developers and report challenges related to building first versions of ML models, especially around the early stages of exploration and experimentation (e.g., feature engineering, model training). They describe the process of building models as "magic"—similarly echoed by Lee et al. [50] when analyzing ML projects from Github—with unique practices of debugging data in addition to code. Serban et al. [88] conduct a survey of practitioners and list 29 software engineering practices for ML, such as "Use Continuous Integration" and "Peer Review Training Scripts." Muiruri et al. [64] interview Finnish engineers and investigate technical challenges and ML-specific tools in the ML lifecycle. Amershi et al. [3] identify challenges such as hidden feedback loops and component entanglement through their interviews with scientists, engineers, and managers at Microsoft. They broadly discuss strategies to integrate support for ML development into traditional software infrastructure, such as end-to-end pipeline support from data engineers and educational conferences for employees. Our work expands on the software engineering for ML ecosystem by considering human-centered, operational requirements for ML deployments, e.g., over time, as MLEs are introduced to ML pipelines that are unfamiliar to them, or as customer or product

requirements change. Unlike Amershi et al., we focus on MLEs, who are responsible for maintaining ML pipeline performance. We also interview practitioners across companies and applications: we provide new and specific examples of ML engineering practices to sustain ML pipelines as software and categorize these practices around a broader human-centered workflow.

The data management, software engineering, and CSCW communities have proposed various software tools for ML workflows. For example, some tools manage data provenance and training context for model debugging purposes [7, 20, 28, 67]. Others help ensure reproducibility while iterating on different ideas [30, 39, 90]. With regards to validating changes in *production* systems, some researchers have studied CI (Continuous Integration) for ML and proposed preliminary solutions—for example, `ease.ml/ci` streamlines data management and proposes unit tests for overfitting [1], and some papers introduce tools to perform validation and monitoring in production ML pipelines [8, 38, 86]. Our work is complementary to existing literature on this tooling; we do not explicitly ask interviewees questions about tools, nor do we propose any tools. We focus on behavioral practices of MLEs.

2.5 MLOps Practices and Challenges

The traditional software engineering literature describes the need for DevOps, a combination of software *dev*elopers and *op*erations teams, to streamline the process of delivering software in organizations [17, 52, 55, 56]. Similarly, the field of *MLOps*, or DevOps principles applied to machine learning, has emerged from the rise of machine learning (ML) application development in software organizations. MLOps is a nascent field, where most existing papers give definitions and overviews of MLOps, as well as its relation to ML, software engineering, DevOps, and data engineering [22, 35, 44, 58, 81, 89, 91, 98–100]. Some work in MLOps attempts to characterize a production ML lifecycle; however, there is little consensus. Symeonidis et al. [98] discuss a lifecycle of data preparation, model selection, and model productionization, but other literature reviews [22, 53] and guides on best practices drawing from authors' experiences [59] conclude that, compared to software engineering, there is not yet a standard ML lifecycle, with consensus from researchers and industry professionals. While standardizing an ML lifecycle across different roles (e.g., scientists, researchers, business leaders, engineers) might be challenging, characterizing a workflow specific to a certain role could be more tractable.

Several MLOps papers present case studies of productionizing ML within specific organizations and the resulting challenges. For example, adhering to data governance standards and regulation is difficult, as model training is data-hungry by nature [5, 25]. Garg et al. [22] discuss issues in continuous end-to-end testing (i.e., continuous integration) because ML development involves changes to datasets and model parameters in addition to code. To address such challenges, other MLOps papers have surveyed the proliferating number of industry-originated tools in MLOps [53, 80, 83, 98]. MLOps tools can help with general pipeline management, data management, and model management [80]. The surveys on tools motivate understanding how MLEs use such tools, to see if there are any gaps or opportunities for improvement.

Prior work in this area—primarily limited to literature reviews, surveys, case studies, and vision papers—motivates research in understanding the *human-centered* workflows and pain points in MLOps. Some MLOps work has interviewed people involved in the production ML lifecycle: for example, Kreuzberger et al. [44] conduct semi-structured interviews with 8 experts from different industries spanning different roles, such as AI architect and Senior Data Platform Engineer, and uncover a list of MLOps principles such as CI/CD automation, workflow orchestration, and reproducibility, as well as an organizational workflow of product initiation, feature engineering, experimentation, and automated ML workflow pipeline. While Kreuzberger et al. [44] explicitly

RR	Id	Role	Org Size	Application	Yrs Xp	Site	Highlights
1	Lg1	MLE Mgr.	Large	Autonomous vehicles	5-10	US-West	high velocity experimentation; scenario testing
1	Md1	MLE	Medium	Autonomous vehicles	5-10	US-West	pipeline-on-a-schedule; copy-paste anomalies
1	Sm1	MLE	Small	Computer hardware	10-15	US-West	exploratory data analysis; AB Testing; SLOs
1	Md2	MLE	Medium	Retail	5-10	US-East	retraining cadence; adaptive test data; feedback delay
1	Lg2	MLE Mgr.	Large	Ads	5-10	US-West	ad click count; model ownership; keeping GPUs warm
1	Lg3	MLE	Large	Cloud computing	10-15	US-West	bucketing / binning; SLOs; hourly batched predictions
2	Sm2	MLE	Small	Finance	5-10	US-West	F1-score ; retraining cadence; progressive validation
2	Sm3	MLE	Small	NLP	10-15	Intl	triage queue; fallback models; false-positive rate
2	Sm4	MLE	Small	OCR + NLP	5-10	Intl	human annotators; word2vec; airflow
3	Md3	MLE Mgr.	Medium	Banking	10-15	US-West	human annotators; institutional knowledge; revenue
3	Lg4	MLE	Large	Cloud computing	10-15	US-West	online inference; pipeline-on-a-schedule; fallback models
3	Sm5	MLE	Small	Bioinformatics	5-10	US-West	model fine-tuning; someone else's features
4	Md4	MLE	Medium	Cybersecurity	10-15	US-East	model-per-customer; join predictions w/ ground truth
4	Md5	MLE	Medium	Fintech	10-15	US-West	retraining cadence; dropped special characters
5	Sm6	MLE	Small	Marketing and analytics	5-10	US-East	human annotators; label quality; adaptive test data
5	Md6	MLE	Medium	Website builder	5-10	US-East	SLOs; poor documentation; data validation
6	Lg5	MLE	Large	Recommender systems	10-15	US-West	pipeline-on-a-schedule; SLOs; progressive validation
6	Lg6	MLE Mgr.	Large	Ads	10-15	US-West	fallback models; on-call rotations; scale

Table 1. The table provides an anonymized description of interviewees from different sizes of companies, roles, years of experience, application areas, and their code attributions. The interviewees hail from a diverse set of backgrounds. **RR** denotes recruitment round. **Highlights** refers to key codes (i.e. from the code system in Fig. 2) attached to that participant's transcript.

interview professionals from different roles to understand shared patterns between their workflows—in fact, only two of the eight interviewees have "machine learning engineer" or "deep learning engineer" in their job titles, our work complements their findings by focusing only on self-declared ML engineers responsible for repeated-use models in production and uncovering strategies they use to sustain model performance. As such, we uncover and present a different workflow—one centered around ML engineering. To the best of our knowledge, we are the first to study the human-centered MLOps workflow from ML engineers' perspectives.

3 METHODS

Upon securing approval from our institution's review board, we conducted an interview study of 18 ML Engineers (MLEs) across various sectors. Our approach mirrored a zigzag model common to Grounded Theory, with alternating phases of fieldwork and in-depth coding and analysis, directing further rounds of interviews [15]. The constant comparative method helped iterate and refine our categories and overarching theory. Consistent with qualitative research standards, theoretical saturation is generally recognized between 12 to 30 participants, particularly in more uniform populations [26]. By our 16th interview, prevalent themes emerged, signaling that saturation was attained. Later interviews confirmed these themes.

Our goal in conducting this study was to develop better tools for ML deployment, broadly targeting monitoring, debugging, and observability issues. Our study was an attempt at identifying key challenges and open opportunities in the MLOps space. This study therefore stems from our collective desire to enrich our understanding of our target community and offer valuable insights into best practices in ML engineering and data science. Subsequent sections delve into participant recruitment (Subsection 3.1), our interview structure (Subsection 3.2), and our coding and analysis techniques (Subsection 3.3).

3.1 Participant Recruitment & Selection

We recruited individuals who were responsible for the development, regular retraining, monitoring and deployment of ML models *in production*. A description of the 18 MLEs is shown in Table 1.

The MLEs interviewed varied in their educational backgrounds, years of experience, roles, team size, and work sector. Recruitment was conducted in rounds over the course of an academic year (2021-2022). Our recruitment strategy was underpinned by a deliberate, iterative process that built upon the insights from each round. The primary goal was to cultivate a representative sample that captured the rich diversity of Machine Learning Engineers (MLEs) across various dimensions.

3.1.1 Initial Recruitment: Relying on Professional Networks. In the initial recruitment round (RR=1), we leaned heavily on the established professional networks of our faculty co-authors. This approach, while convenient and efficient, resulted in a sample that was geographically skewed towards the US-West. It also led to a greater representation from larger organizations, as well as certain sectors like Autonomous Vehicles and Cloud Computing. This initial cohort provided valuable insights but also highlighted the potential biases and gaps in our sample.

3.1.2 Course Correction: Diversifying the Sample. Recognizing the need for a more representative and diversified sample, our strategy in subsequent rounds shifted. Specifically for RR=2, we made a concerted effort to engage candidates outside our immediate professional networks and particularly targeted those at smaller companies. This shift in approach was operationalized by posting recruitment advertisements and flyers in various online communities. Prospective participants who responded to our outreach underwent a screening process for the same inclusion criteria mentioned previously. Their professional backgrounds and affiliations were verified through platforms such as professional websites, LinkedIn, and online portfolios. As a result, we observed a stronger representation from domains like NLP and Finance.

3.1.3 Building a Representative Sample: Iterative Refinement. Each recruitment round served as a feedback loop, informing the strategy for the subsequent round. As patterns emerged from our data analysis, we fine-tuned our recruitment focus to fill identified gaps. This iterative process ensured that, over time, our sample grew to be more balanced in terms of roles, experience, organization sizes, sectors, and geographical locations. By employing this iterative recruitment strategy, we believe our study encapsulates a comprehensive cross-section of the MLE community, offering insights that are both deep and broad.

In each round, between three to five candidates were reached by email and invited to participate. We relied on our professional networks and open calls posted on MLOps channels in Discord[1], Slack[2], and Twitter to compile a roster of candidates. The roster was incrementally updated roughly after every round of interviews, integrating information gained from the concurrent coding and analysis of transcripts (Section 3.3). Recruitment rounds were repeated until we reached saturation on our findings [65].

3.2 Interview Protocol

With each participant, we conducted semi-structured interviews over video call lasting 45 to 75 minutes each. Over the course of the interview, we asked descriptive, structural, and contrast questions abiding by ethnographic interview guidelines [92]. The questions are listed in Appendix A. Specifically, our questions spanned six categories: i) the type of ML tasks they work on, ii) the approaches used for building or tuning models, iii) the transition from development to production, iv) how they evaluate their models before deployment, v) how they monitor their deployed models, and vi) how they respond to issues or bugs. Participants received and signed a consent form before the interview, and agreed to participate free of compensation. As per our agreement, we automatically transcribed the interviews using Zoom software. In the interest of privacy and

[1]https://discord.com/invite/Mw77HPrgjF
[2]mlops-community.slack.com

(1) **Tasks**
  (a) Data collection, cleaning & labeling: *human annotators*, *exploratory data analysis*
  (b) Embeddings & feature engineering: *normalization*, *bucketing / binning*, *word2vec*
  (c) Data modeling & experimentation: *accuracy*, *F1-score*, *precision*, *recall*
  (d) Testing: *scenario testing*, *AB testing*, *adaptive test-data*
(2) **Biz/Org Management**
  (a) Business focus: *service level objectives*, *ad click count*, *revenue*
  (b) Teams & collaboration: *institutional knowledge*, *on-call rotations*, *model ownership*
  (c) Process maturity indicators: *pipeline-on-a-schedule*, *fallback models*, *model-per-customer*
(3) **Operations**
  (a) CI/CD: *artifact versioning*, *multi-staged deployment*, *progressive validation*
  (b) Data ingestion & integration: *automated featurization*, *data validation*
  (c) Model retraining: *distributed training*, *retraining cadence*, *model fine-tuning*
  (d) Prediction serving: *hourly batched predictions*, *online inference*
  (e) Live monitoring: *false-positive rate*, *join predictions w/ ground truth*
(4) **Bugs & Pain Points**
  (a) Bugs: *data leakage*, *dropped special characters*, *copy-paste anomalies*
  (b) Pain points: *big org red tape*, *performance regressions*, *label quality*, *scale*
  (c) Anti-patterns: *muting alerts*, *keeping GPUs warm*, *waiting it out*
  (d) Known challenges: *data drift*, *feedback delay*, *class imbalance*, *sensor divergence*
  (e) Missing context: *someone else's features*, *poor documentation*, *much time has passed*
(5) **Systems & Tools**
  (a) Metadata layer: *Huggingface*, *Weights & Biases*, *MLFlow*, *TensorBoard*, *DVC*
  (b) Unit layer: *PyTorch*, *TensorFlow*, *Jupyter*, *Spark*
  (c) Pipeline layer: *Airflow*, *Kubeflow*, *Papermill*, *DBT*, *Vertex AI*
  (d) Infrastructure layer: *Slurm*, *S3*, *EC2*, *GCP*, *HDFS*

Fig. 2. Abridged code system: A distilled representation of the evolved code system resulting from our qualitative study, capturing the primary tasks, organizational aspects, operational methodologies, challenges, and tools utilized by Machine Learning Engineers.

confidentiality, we did not record audio or video of the interviews. Transcripts were redacted of personally identifiable information before being uploaded to a secured drive in the cloud.

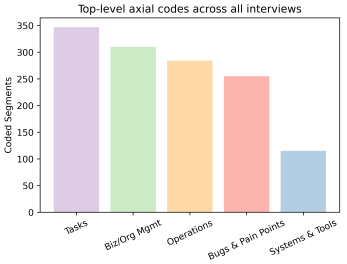
3.3 Transcript Coding & Analysis

We employed grounded theory to systematically analyze our interview transcripts. Grounded theory is a robust methodology focused on iterative data collection and analysis for theory discovery [12, 95]. One of its key features is the seamless integration of data collection and analysis, aiming to identify emerging themes and concepts through a constant review of transcripts. To facilitate this, we used open and axial coding techniques and employed MaxQDA software for qualitative analysis.

Axial coding serves as a structured method for categorizing and organizing codes into a hierarchical system. During the study, the authors convened weekly to review previous interviews and refine the evolving coding structure. By progressively grouping or merging existing codes, we developed a hierarchical tree of concepts to effectively navigate and interpret our interview data. Ultimately, we identified five top-level codes, each encapsulating core themes and concepts of our study. These codes and their respective sub-categories are detailed in the abridged code system (see

Participant	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
p1	0	0	4	8	4	7	11	5	6	9	6	2	6	5	10	7	2	2	4	1	1	1	3	3	1	1			
p2	0	0	5	8	2	8	10	1	0	4	7	3	6	4	1	4	3	7	1	3	1	2	1						
p3	0	3	3	0	4	3	4	0	0	0	0	1	0	1	3	1	1	1	6	2									
p5	2	6	1	6	4	8	4	8	4	6	7	2	4	2	7	5	4	4	3	3	4	1	1						
p6	2	1	0	4	7	7	4	3	7	4	0	2	5	8	2	6	1	1	3	2	5	4	8	1	3	6	4	5	5
p7	8	9	2	2	3	3	2	5	8	6	5	7	2	1	1	3	1	0	2	3	2	2	1	2	0	1	2		
p8	2	5	1	2	7	1	9	4	3	3	4	0	3	2	6	4	2	3	2	3									
p10	0	2	4	5	6	9	3	2	2	1	10	5	4	6	5	3	4	2	3	2	4	6	3	0	1	1			
p11	1	4	7	2	2	0	1	0	2	0	3	2	1	1	2	2	2	4	2	2	4	0	0	0	0	1			
p12	0	2	1	2	2	5	1	1	1	2	0	2	1	1	1	2	3	1											
p13	2	2	1	1	4	1	1	2	2	2	2	2	3	3	2	2	2	1	1	2	1	2							
p14	3	6	6	4	0	3	6	5	7	12	4	7	3	5	7	3	3	6	3	3	3	3	0	2	3	3	1	2	2
p15	0	1	0	4	3	5	1	1	3	4	4	0	4	1	2	1	2	1	6	7	0	1	2	1	2	2	2	0	1
p16	0	4	3	3	5	6	2	1	1	4	1	2	4	2	5	2	3	4	3	3	6	1	1	3	0	0	0	0	1
p17	2	0	2	2	5	6	6	2	5	5	6	5	6	3	5	4	4	7	2	0	2	3	1	1	2	2	1		
p18	14	9	8	12	5	6	8	8	5	3	13	3	7	4	3	5	5	8	4	5	0	4							
p19	0	4	3	4	4	4	6	0	4	3	4	1	1	4	0	0	0	0	3	1	1	4	1						

(a) Color-coded Overview of Transcripts



(b) Color Legend

Fig. 3. Visual summary of coded transcripts. The x-axis of (a), the color-coded overview, corresponds to a segment (or group) of transcript lines, and the number in each cell is the code's frequency for that transcript segment and for that participant. Segments are blank after the conclusion of each interview, and different interviews had different time duration. Each color in (a) is associated with a top-level axial code from our interview study, and presented in the color legend (b). The color legend also shows the frequency of each code across all interviews.

Figure 2). As illustrated in Figure 3, we allocated roughly equal time to each main theme, which correspondingly informed our findings. The themes relate to our findings as follows:

(1) **Tasks** refers to activities that are routinely performed by ML engineers. The analysis of code segments descended from *tasks*, and its decomposition into constituent parts, culminated in the creation of the MLOps workflow (Figure 1), and is instrumental in the structure and presentation of Section 4 (*Findings*).
(2) **Biz/Org (Business-Organizational) Management** refers to modes of interaction between MLEs and their co-workers or managers, and between MLEs and customers or other stakeholders. Relevant sub-codes form the theoretical basis for Section 4.2.2 (*Collaboration*) and Section 4.3.2 (*Product Metrics*).
(3) **Operations** refers to repeatable work that must be performed regularly and consistently for the continued operation of the business. *Operations* is the "Ops" in MLOps. Relevant sub-codes form the theoretical basis for Section 4.1.1 (*Pipeline Automation*).
(4) **Bugs & Pain Points** refers to failure modes encountered at any stage in the MLOps workflows, MLE complaints generally, and author-noted anti-patterns. These are discussed in Monitoring and Response (Section 4.4).
(5) **Systems & Tools** refers to storage and compute infrastructure, programming languages, ML training frameworks, experiment execution frameworks, and other tools or systems that MLEs use in their MLOps work. We discuss implications for tool design in Section 5.2. We include a table of common tools referenced by interviewees in Appendix A.

4 SUMMARY OF FINDINGS

Going into the interview study, we assumed the workflow of human-centered tasks in the production ML lifecycle was similar to the production data science workflow presented by Wang et al. [102], which is a loop consisting of the following:

(1) **Preparation**, spanning data acquisition, data cleaning and labeling, and feature engineering,
(2) **Modeling**, spanning model selection, hyperparameter tuning, and model validation, and
(3) **Deployment**, spanning model deployment, runtime monitoring, and model improvement.

From our interviews, we found that the repeated-use production ML workflow that ML engineers engage in differs slightly from Wang et al. [102]. As preliminary research papers defining and providing reference architectures for MLOps have pointed out, operationalizing ML brings new requirements to the table, such as the need for teams, not just individual people, to understand, sustain, and improve ML pipelines and systems over time [22, 44, 98]. In the pipelines that our interviewees build and supervise, most technical components are automated—e.g., data preprocessing jobs run on a schedule, and models are typically retrained regularly on fresh data. We found the ML engineering workflow to revolve around the following stages (Figure 1):

(1) **Data Preparation**, which includes scheduled data acquisition, cleaning, labeling, and transformation,
(2) **Experimentation**, which includes both data-driven and model-driven changes to increase overall ML performance, and is typically measured by metrics such as accuracy or mean-squared-error,
(3) **Evaluation & Deployment**, which consists of a model retraining loop which periodically rolls out a new model—or offline batched predictions, or another proposed change—to growing fractions of the population, and
(4) **Monitoring & Response**, which supports the other stages via data and code instrumentation (e.g., tracking available GPUs for experimentation or the fraction of null-valued data points) and dispatches engineers and bug fixes to identified problems in the pipeline.

For each stage, we identified human-centered practices from the *Tasks*, *Biz/Org Management*, and *Bugs & Pain Points* codes, and drew on *Operations* codes for automated practices (refer to Section 3.3 for a description of these codes). An overview of findings for each workflow stage can be found in Table 2.

The following subsections organize our findings around the four stages of MLOps. We begin each subsection with a quote that stood out to us and conversation with prior work; then, in the context of what is automated, we discuss common human-centered practices and pain points.

4.1 Data Preparation

"It takes exponentially more data to keep getting linear increases in performance." –Lg1

Data preparation is the process of constructing "well-structured, complete datasets" for data scientists [66]. Data preparation activities consist of collection, wrangling, and cleaning and are known to be challenging, often taking up to 80% of practitioners' time [36, 102]. This tedious process encourages larger organizations to have dedicated teams of data engineers to manage data preparation [37]. Like Amershi et al. [3], we observed that mature ML organizations automated data preparation through dedicated teams as much as possible (Lg1, Lg2, Lg3, Sm3, Md3, Sm6, Md6, Lg5, Lg6). As a result, the MLEs we interviewed spent a smaller fraction of their time on data preparation, collaborating instead with data engineering teams. We first discuss pipeline automation to provide key context for the work of MLEs. Then, we mention two key challenges MLEs face: ensuring labeling quality at scale and coping with feedback delays.

4.1.1 ***Pipelines automatically run on a schedule.*** Unlike data science, where data preparation is often ad-hoc and interactive [37, 66], we found that data preparation in production ML is batched and more narrowly restricted, involving an organization-wide set of steps running at a predefined cadence. In interviews, we found that preparation pipelines were defined by Directed Acyclic Graphs, or DAGs, which ran on a schedule (e.g., daily). Each DAG node corresponded to a particular task, such as ingesting data from a source or cleaning a newly ingested partition of data. Each DAG edge corresponded to a dataflow dependency between tasks. While data engineers were primarily responsible for the end-to-end execution of data preparation DAGs, MLEs interfaced with these

Stage	Description	Non-Automated Challenges
Data Preparation	Collection, wrangling, and cleaning pipelines run on a schedule	- Ensuring label quality at scale (Section 4.1.2) - Handling feedback or ground-truth delays (Section 4.1.3)
Experimentation	Prototyping ideas to improve end-to-end ML pipeline performance by iterating on datasets, model architectures, or both	- Managing the underlying software or code for data-centric experiments (Section 4.2.1) - Engaging in cross-team collaboration (Section 4.2.2) - Manually and thoughtfully identifying promising experiment configurations (Section 4.2.3)
Evaluation and Deployment	Pushing experimental changes to small, then large fractions of users, evaluating at every step	- Continuously updating dynamic validation datasets for future experiments (Section 4.3.1) - Using product metrics for evaluation (Section 4.3.2)
Monitoring and Response	Supervising live ML pipeline performance and minimizing pipeline downtime	- Tracking and investigating data quality alerts (Section 4.4.1) - Managing pipeline "jungles" of models and hard-coded rules (Section 4.4.2) - Debugging a heavy-tailed distribution of errors (Section 4.4.3)

Table 2. Overview of challenging activities that ML engineers engage in for each stage in their workflow. While each stage relies on automated infrastructure and pipelines, ML engineers still have many difficult manual responsibilities.

DAGs by loading select outputs (e.g., clean data) or by extending the DAG with additional tasks, e.g. to compute new features (Md1, Lg2, Lg3, Sm4, Md6, Lg6).

In many cases, automated tasks relating to *ML models*, such as model inference (e.g., generating predictions with a trained model) and retraining, were executed in the same DAG as data preparation tasks (Lg1, Md1, Sm2, Md4, Md5, Sm6, Md6, Lg5). ML engineers included retraining as a node in the data preparation DAG for simplicity: as new data becomes available, a corresponding model is automatically retrained. Md4 mentioned automatically retraining the model every day so model performance would not suffer for longer than a day:

> *Why did we start training daily? As far as I'm aware, we wanted to start simple—we could just have a single batch job that processes new data and we wouldn't need to worry about separate retraining schedules. You don't really need to worry about if your model has gone stale if you're retraining it every day.*

Retraining cadences ranged from hourly (Lg5) to every few months (Md6) and were different for different models within the same organization (Lg1, Md4). None of the participants interviewed reported any scientific procedure for determining the pipeline execution cadence. For example, Md5 said that "the [model retraining] cadence was just like, finger to the wind." Cadences seemed to be set in a way that streamlined operations for the organization in the easiest way. Lg5 mentioned that "retraining took about 3 to 4 hours, so [they] matched the cadence with it such that as soon as [they] finished any one model, they kicked off the next training [job]." Engineers reported an inability to retrain unless they had fresh and labeled data, motivating their organizations to set up dedicated teams of annotators, or hiring crowd workers, to operationalize labeling of live data (Lg1, Sm3, Sm4, Md3, Sm6).

4.1.2 ***MLEs ensure label quality at scale.*** Although it is widely recognized that model performance improves with more labels [82]—and there are tools built especially for data labeling [79, 111]—our interviewees cautioned that the quality of labels can degrade as they try to label more and more data. Md3 said:

> *No matter how many labels you generate, you need to know what you're actually labeling and you need to have a very clear human definition of what they mean.*

In many cases, ground truth must be *created*, i.e., the labels are what a practitioner "thinks up" [66]. When operationalizing this practice, MLEs faced problems. For one, Sm3 spoke about how expensive it was to outsource labeling. Moreover, labeling conflicts can erode trust in data quality, and slow ML progress [73, 85]: When scaling up labeling—through labeling service providers or analysts within the organization—MLEs frequently found disagreements between different labelers, which would negatively impact quality if unresolved (Sm3, Md3, Sm6). At their organization, Sm3 mentioned that there was a human-in-the-loop labeling pipeline that both outsourced large-scale labeling and maintained an internal team of experts to verify the labels and resolve errors manually. Sm6 described a label validation pipeline for outsourced labels that itself used ML models for estimating label quality.

4.1.3 ***Feedback delays can disrupt pipeline cadences.*** In many applications, today's predictions are tomorrow's training data, but many participants said that ground-truth labels for live predictions often arrived after a delay, which could vary unpredictably (e.g., human-in-the-loop or networking delays) and thus caused problems for knowing real-time performance or retraining regularly (Md1, Sm2, Sm3, Md5, Md6, Lg5). This is in contrast to academic ML, where ground-truth labels are readily available for ML practitioners to train models [71]. Participants noted that the impact on models was hard to assess when the ground truth involved live data—for example, Sm2 felt strongly about the negative impact of feedback delays on their ML pipelines:

> *I have no idea how well [models] actually perform on live data. Feedback is always delayed by at least 2 weeks. Sometimes we might not have feedback...so when we realize maybe something went wrong, it could [have been] 2 weeks ago.*

Feedback delays contribute to "data cascades," or compounding errors in ML systems over time [85]. Sm3 mentioned a 2-3 *year* effort to develop a human-in-the-loop pipeline to manually label live data as frequently as possible to side-step feedback delays: "you want to come up with the rate at which data is changing, and then assign people to manage this rate roughly". Sm6 said that their organization hired freelancers to label "20 or so" data points by hand daily. Labeling was then considered a task in the broader preparation pipeline that ran on a schedule (Section 4.1.1).

4.2 Experimentation

> *"You want to see some degree of experimental thoroughness. People will have principled stances or intuitions for why things should work. But the most important thing to do is achieve scary high experimentation velocity...Number one [Key Performance Indicator] is rate of experimentation." (Lg1*

While most prior work studying the data science and MLOps workflows includes *modeling* as an explicit step in the workflow [96, 102, 106, 110], we found that iterating on model ideas and architectures is only part of a broader "experimentation" step. This is because in many production ML pipelines, MLEs can focus on tuning or improving existing models through data-centric development, and *modeling* in a data science sense is only necessary when the company wishes to expand its service offerings or grow its ML capabilities. In fact, many of our interviewees did not build the initial model in the pipeline that their organization assigned them to work on, so their

goal wasn't necessarily to train more models. As an example, Md6 said, "some of our models have been around for, like, 6 or 7 years." Garg et al. [22] also call this workflow step "experimentation" instead of "modeling" in their MLOps lifecycle overview, and we expand on this finding in our paper by relating it to collaboration and data-driven exploration, as well as MLE reservations toward experiment automation or AutoML.

4.2.1 ***MLEs find it better to innovate on the data than the model.*** Holstein et al. [31] mention that it is challenging for practitioners to determine where to focus experimentation efforts—they could try "switching to a different model, augmenting the training data in some way, collecting more or different kinds of data, post-processing outputs, changing the objective function, or something else." Our interviewees recommended focusing on experiments that provided additional context to the model, typically via new features, to get the biggest gains (Lg2, Lg3, Md3, Lg4, Md4, Sm6, Md6, Lg5, Lg6). Lg5 said:

> *I'm focusing my energy these days on signals and feature engineering because even if you keep your code static and the model static, it would definitely help you with getting better model performance.*

In a concurring view, Md3 adds:

> *I'm gonna start with a [fixed] model because it means faster iterations. And often—like most of the time empirically—it's going to be something in our data that we can use to push the boundary [...] obviously, it's not a dogmatic "we will never touch the model" but it shouldn't be our first move.*

Interestingly, older work claims that iterating on the model is often more fruitful than iterating on the data [74], but this could be because ML modeling libraries weren't as mature as they are now. Recent work has also identified the importance of data-centric experimentation in production ML deployments [66, 71, 79, 85]. Md6 mentioned that most ML projects at their organization centered around adding new features. Md4 mentioned that one of their current projects was to move feature engineering pipelines from Scala to SparkSQL (a language more familiar to ML engineers and data scientists), so experiment ideas could be coded and validated faster.

When asked how they managed the underlying software or code for data-centric experiments, interviewees emphasized the importance of keeping their code changes as small as possible for multiple reasons, including faster code review, easier validation, and fewer merge conflicts (Md1, Lg2, Lg3, Sm4, Md3, Lg5, Lg6). This is in line with good software engineering practices [3]. Additionally, changes in large organizations were primarily made in configuration (config) files instead of main application code (Lg1, Md1, Lg2, Sm4, Lg4, Lg6): instead of editing parameters directly in a Python model training script, MLEs preferred to edit a config file of parameters (e.g., JSON or YAML), and would feed the config file to the model training script. When larger changes were necessary, especially changes touching the language layer (e.g. changing PyTorch or TensorFlow architectures), MLEs would fork the code base and made their edits in-place (Md2, Lg3). Although forking repositories can be a high-velocity shortcut, absent streamlined merge procedures, this can lead to a divergence in versions and accumulation of technical debt. Lg3 highlighted the tension between experiment velocity and strict software engineering discipline:

> *I used to see a lot of people complaining that model developers don't follow software engineering. At this point, I'm feeling more convinced that it's not because they're lazy. It's because [software engineering is] contradictory to the agility of analysis and exploration.*

4.2.2 ***Feature engineering is social and collaborative.*** Prior work has stressed the importance of collaboration in data science projects, often lamenting that technical tasks happen in silos [46, 71, 85, 102]. Our interviewees similarly believed cross-team collaboration was critical for successful

experiments. Project ideas, such as new features, came from or were validated early by domain experts: data scientists and analysts who had already performed a lot of exploratory data analysis. Md4 and Md6 independently recounted successful project ideas that came from asynchronous conversations on Slack: Md6 said, "I look for features from data scientists, [who have ideas of] things that are correlated with what I'm trying to predict." We found that organizations explicitly prioritized cross-team collaboration as part of their ML culture. Md3 said:

> *We really think it's important to bridge that gap between what's often, you know, a [subject matter expert] in one room annotating and then handing things over the wire to a data scientist—a scene where you have no communication. So we make sure there's both data science and subject matter expertise representation [in our meetings].*

To foster a more collaborative culture, Sm6 discussed the concept of "building goodwill" with other teams through tedious tasks that weren't always explicitly a part of company plans: "Sometimes we'll fix something [here and] there to like build some goodwill, so that we can call on them in the future...I do this stuff [to build relationships], not because I'm really passionate about doing it."

4.2.3 ***MLEs like having manual control over experiment selection.*** One challenge that results from fast exploration is having to manage many experiment versions [44, 102]. MLEs are happy to delegate experiment tracking and execution work to ML experiment execution frameworks, such as Weights & Biases[3], but prefer to choose subsequent experiments themselves. To be able to make informed choices of subsequent experiments to run, MLEs must maintain awareness of what they have tried and what they haven't (Lg2 calls it the "exploration frontier"). As a result, there are limits to how much automation MLEs are willing to rely on for experimentation, a finding consistent with results from Xin et al. [108]. Lg2 mentioned the phrase "keeping GPUs warm" to characterize a perceived anti-pattern that wastes resources without a plan:

> *One thing that I've noticed is, especially when you have as many resources as [large companies] do, that there's a compulsive need to leverage all the resources that you have. And just, you know, get all the experiments out there. Come up with a bunch of ideas; run a bunch of stuff. I actually think that's bad. You can be overly concerned with keeping your GPUs warm, [so much] so that you don't actually think deeply about what the highest value experiment is.*

In executing experiment ideas, we noticed a tradeoff between a guided search and random search. Random searches were more suited to parallelization—e.g., hyperparameter searches or ideas that didn't depend on each other. Although computing infrastructure could support many different experiments in parallel, the cognitive load of managing such experiments was too cumbersome for participants (Lg2, Sm4, Lg5, Lg6). Rather, participants noted more success when pipelining learnings from one experiment into the next, like a guided search to find the best idea (Lg2, Sm4, Lg5). Lg5 described their ideological shift from random search to guided search:

> *Previously, I tried to do a lot of parallelization. If I focus on one idea, a week at a time, then it boosts my productivity a lot more.*

By following a guided search, engineers are, essentially, significantly pruning a large subset of experiment ideas without executing them. While it may seem like there are unlimited computational resources, the search space is much larger, and developer time and energy is limited. At the end of the day, experiments are human-validated and deployed. Mature ML engineers know their personal tradeoff between parallelizing disjoint experiment ideas and pipelining ideas that build on top of each other, ultimately yielding successful deployments.

[3]https://wandb.ai/

4.3 Evaluation and Deployment

> *"We don't have a good idea of how the model is going to behave in production until production." (Lg3)*

After the experimentation phase, when MLEs have identified a change they want to make to the ML pipeline (e.g., adding a new feature), they need to evaluate it and deploy it to production. Prior work that prescribes frameworks for MLOps typically separates evaluation and deployment into two different stages [48, 96, 98], but we combine them into one step because they are tightly intertwined, with deployments spanning long periods of time and evaluations happening multiple times during deployment.

Prior work describes evaluation as an "offline," automated process that happens at training time: a small portion of the training dataset is held out, and the model should achieve high accuracy on this held-out set [71, 106]. Recent related work in MLOps claims that evaluation and deployment are highly amenable to automation [22, 59]. As such, we also originally hypothesized that evaluation and deployment could be automated—once validated, an engineer could simply create a new task in their DAG to retrain the model on a cadence (Section 4.1.1).

As expected, engineers did automatically validate and codify their changes into DAGs to retrain models on a schedule. However, they also manually supervised the deployment over a long period of time, evaluating throughout the time frame. Amershi et al. [3] state that software teams "flight" changes or updates to ML models, often by testing them on a few cases prior to live deployment. Our work provides further context into the evaluation and deployment process for production ML pipelines: we found that several organizations, particularly those with many customers, employed a *multi-stage deployment process* for new models or model changes, progressively evaluating at each stage (Sm1, Lg2, Lg3, Sm2, Sm3, Lg4, Md5, Md6, Lg5, Lg6). As such, we combine evaluation and deployment into one step, rather than separating the process into evaluation followed by deployment as other papers do [96, 98]. Lg3 described the multi-staged deployment process as follows:

> *We have designated test clusters, [stage 1] clusters, [stage 2] clusters, then the global deployment [to all users]. The idea here is you deploy increasingly along these clusters, so that you catch problems before they've met customers.*

Each organization had different names for its stages (e.g., test, dev, canary, staging, shadow, A/B) and different numbers of stages in the deployment process (usually between one and four). The stages helped invalidate models that might perform poorly in full production, especially for brand-new or business-critical cases. Occasionally, organizations had an offline "sandbox" stage preceding deployment to any fraction of customers—for example, Md5 described a sandbox where they could stress-test their chatbot product:

> *You can pretend to be a customer and say all sorts of offensive things, and see if the model will say cuss words back at you, or other sorts of things like that.*

Although the model retraining process was automated, we find that MLEs personally reviewed validation metrics and manually supervised the promotion from one stage to the next. They had oversight over every evaluation stage, taking great care to manage complexity and change over time: specifically, changes in data, product and business requirements, users, and teams within organizations. We discuss two human-centered practices: maintaining dynamic datasets and evaluating performance in the context of the product or broader organizational value.

4.3.1 ***MLEs continuously update dynamic validation datasets.*** Many engineers reported processes to analyze live failure modes and update the validation datasets to prevent similar failures

from happening again (Lg1, Md1, Lg2, Lg3, Sm3, Md3, Md5, Sm6, Md6, Lg5). Lg1 described this process as a departure from what they had learned in school:

> *You have this classic issue where most researchers are evaluat[ing] against fixed data sets [...but] most industry methods change their datasets.*

We found that these dynamic validation sets served two purposes: (1) the obvious goal of making sure the validation set stays current with live data as much as possible, given new knowledge about the problem and general shifts in the data distribution, and (2) the more specific goal of addressing localized shifts within sub-populations, such as low accuracy for minority groups. The challenge with (2) is that many sub-populations are often overlooked, or they are discovered post-deployment [31]. In response, Md3 discussed how they systematically bucketed their data points based on the model's error metrics and created validation sets for each under-performing bucket:

> *Some [of the metrics in my tool] are standard, like a confusion matrix, but it's not really effective because it doesn't drill things down [into specific subpopulations that users care about]. Slices are user-defined, but sometimes it's a little bit more automated. [During offline evaluation, we] find the error bucket that [we] want to drill down, and then [we] either improve the model in very systematic ways or improve [our] data in very systematic ways.*

Rather than follow a proactive approach of constructing different failure modes in an offline validation phase like Md3 did, Sm3 offered a reactive strategy of spawning a new dataset for each observed live failure: "Every [failed prediction] gets into the same queue, and 3 of us sit down once a week and go through the queue...then our [analysts] collect more [similar] data." This dataset update (or delta) was then merged into the validation dataset, and used for model validation in subsequent rounds. While processes to dynamically update the validation datasets ranged from human-in-the-loop to periodic synthetic data construction (Lg3), we found that higher-stakes applications of ML (e.g., autonomous vehicles), created dedicated teams to manage the dynamic evaluation process. Often, this involved creating synthetic data representative of live failures (Lg1, Lg3, Md4). For example, Lg1 said:

> *What you need to be able to do in a mature MLOps pipeline is go very quickly from user recorded bug, to not only are you going to fix it, but you also have to be able to drive improvements to the stack by changing your data based on those bugs.*

Notwithstanding, participants found it challenging to collect the various kinds of failure modes and monitoring metrics for each mode. Lg6 added, "you have to look at so many different metrics. Even very experienced folks question this process like a dozen times."

4.3.2 ***MLEs use product metrics for validation.*** While prior work discusses how prediction accuracy doesn't always correlate with real-world outcomes [71, 102], it's unclear how to articulate clear and measurable ML goals. Patel et al. [74] discuss how practitioners trained in statistical techniques "felt that they must often manage concerns outside the focus of traditional evaluation metrics." Srivastava et al. [93] note that an increase in accuracy might not improve overall system "compatibility." In our study, we found that successful ML deployments tied their performance to product metrics. First, we found that *prior to initial deployment*, mature ML teams defined a product metric in consultation with other stakeholders, such as business analysts, product managers, or customers (Lg2, Sm2, Md5, Sm6, Md3, Md6, Lg5, Lg6). Examples of product metrics include click-through rate and user churn rate. Md3 felt that a key reason many ML projects fail is that they don't measure metrics that will yield the organization value:

> *Tying [model performance] to the business's KPIs (key performance indicators) is really important. But it's a process—you need to figure out what [the KPIs] are, and frankly I think that's how people should be doing AI. It [shouldn't be] like: hey, let's do these experiments and get cool numbers and show off these nice precision-recall curves to our bosses and call it a day. It should be like: hey, let's actually show the same business metrics that everyone else is held accountable to to our bosses at the end of the day.*

Since product-specific metrics are, by definition, different for different ML models, it was important for engineers to treat the choice of metrics as an explicit step in their workflow and align with other stakeholders to make sure the right metrics were chosen. After agreeing on a product metric, engineers only promoted experiment ideas to later deployment stages if there was an improvement in that metric. Md6 said that every model change in production was validated by the product team: "if we can get a statistically significant greater percentage [of] people to subscribe to [the product], then [we can fully deploy]." Kim et al. [40] also highlight the importance of including other stakeholders (or people in "decision-making seats") in the evaluation process. At each stage of deployment, some organizations placed additional emphasis on important customers during evaluation (Lg3, Sm4). Lg3 mentioned that there were "hard-coded" rules for "mission-critical" customers:

> *There's an [ML] system to allocate resources for [our product]. We have hard-coded rules for mission critical customers. Like at the start of COVID, there were hospital [customers] that we had to save [resources] for.*

Finally, participants who came from research or academia noted that tying evaluation to the product metrics was a different experience. Lg3 commented on their "mindset shift" after leaving academia:

> *I think about where the business will benefit from what we're building. We're not just shipping fake wins, like we're really in the value business. You've got to see value from AI in your organization in order to feel like it was worth it to you, and I guess that's a mindset that we really ought to have [as a community].*

4.4 Monitoring and Response

> *"This data is supposed to have 50 states, there's only 40, what happened to the other 10?" (Md6)*

We found that organizations centered their monitoring and response practices around engineers, much like in the DevOps agile framework, which organizes software development around teams [13]. Prior work has stated that monitoring is critical to MLOps [44, 59, 96, 98], and, broadly, that Agile practices can be useful in supervising production ML [3]. We provide further insight by discussing two specific examples of Agile practices that our interviewees commonly adapted to the ML context. First, Lg3, Lg4, Md4, Sm6, Lg5, and Lg6 described *on-call processes* for supervising production ML models. For each model, at any point in time, some ML engineer would be on call, or primarily responsible for it. Any bug or incident observed (e.g., user complaint, pipeline failure) would receive a ticket, created by the on-call engineer. On-call rotations typically lasted one or two weeks. At the end of a shift, an engineer would create an incident report—possibly one for each bug—detailing major issues that occurred and how they were fixed. Additionally, Lg3, Sm2, Sm4, and Md5 discussed having *Service-Level Objectives (SLOs)*, or commitments to minimum standards of performance, for pipelines in their organizations. For example, a pipeline to classify images could have an SLO of 95% accuracy. A benefit of using the SLO framework for ML pipelines is a clear indication of whether a pipeline is performing well or not—if the SLO is not met, the pipeline is broken, by definition.

Our interviewees stressed the importance of logging data across all stages of the ML pipeline (e.g., feature engineering, model training) to use for future debugging. Monitoring ML pipelines and responding to bugs involved tracking live metrics (via queries or dashboards), slicing and dicing sub-populations to investigate prediction quality, patching the model with non-ML heuristics for known failure modes, and finding in-the-wild failures that could be added to future dynamic validation datasets. While MLEs tried to automate monitoring and response as much as possible, we found that solutions were lacking and required significant human-in-the-loop intervention. Next, we discuss data quality alerts, pipeline jungles, and diagnostics.

4.4.1 ***On-call MLEs track data quality alerts and investigate a fraction of them.*** In data science, data quality is of utmost importance [37, 73]. Prior work has stressed the importance of monitoring data in production ML pipelines [40, 85, 87], and the data management literature has proposed numerous data quality metrics [8, 76, 86]. But what metrics do practitioners actually use, what data do practitioners monitor, and how do they manually engage with these metrics? We found that engineers continuously monitored features for and predictions from production models (Lg1, Md1, Lg3, Sm3, Md4, Sm6, Md6, Lg5, Lg6): Md1 discussed hard constraints for feature columns (e.g., bounds on values), Lg3 talked about monitoring completeness (i.e., fraction of non-null values) for features, Sm6 mentioned embedding their pipelines with "common sense checks," implemented as hard constraints on columns, and Sm3 described schema checks—making sure each data item adheres to an expected set of columns and their types. These checks were automated and executed as part of the larger pipeline (Section 4.1.1).

While off-the-shelf data validation was definitely useful for the participants, many of them expressed pain points with existing techniques and solutions. Lg3 discussed that it was hard to figure out how to trigger alerts based on data quality:

> *Monitoring is both metrics and then a predicate over those metrics that triggers alerts. That second piece doesn't exist—not because the infrastructure is hard, but because no one knows how to set those predicate values...for a lot of this stuff now, there's engineering headcount to support a team doing this stuff. This is people's jobs now; this constant, periodic evaluation of models.*

We also found that employee turnover makes data validation unsustainable (Sm2, Md4, Sm6, Md6, Lg5). If one engineer manually defined checks and bounds for each feature and then left the team, another engineer would have trouble interpreting the predefined data validation criteria. To circumvent this problem, some participants discussed using black-box data monitoring services but lamented that their statistics weren't interpretable or actionable (Sm2, Md4).

Another commonly discussed pain point was *false-positive alerts*, or alerts triggered even when the ML performance is adequate. Engineers often monitored and placed data quality alerts on each feature and prediction (Lg2, Lg3, Sm3, Md3, Md4, Sm6, Md6, Lg5, Lg6). If the number of metrics tracked grew too large, false-positive alerts could become a problem. An excess of false-positive alerts led to fatigue and silencing of alerts, which could miss actual performance drops. Sm3 said "people [were] getting bombed with these alerts." Lg5 shared a similar sentiment, that there was "nothing critical in most of the alerts." The only time there was something critical was "way back when [they] had to actually wake up in the middle of the night to solve it...the only time [in years]." When we asked what they did about the noncritical alerts and how they acted on the alerts, Lg5 said:

> *You typically ignore most alerts...I guess on record I'd say 90% of them aren't immediate. You just have to acknowledge them [internally], like just be aware that there is something happening.*

Seasoned MLEs thus preferred to view and filter alerts themselves, than to silence or lower the alert reporting rate. In a sense, even false-positives can provide information about system health, if the MLE knows how to read the alerts and is accustomed to the system's reporting patterns. When alert fatigue materialized, it was typically when engineers were on-call, or responsible for ML pipelines during a 7 or 14-day shift. Lg6 recounted how on-call rotations were dreaded amongst their team, particularly for new team members, due to the high rate of false-positive alerts. They said:

> *On-call ML engineers freak out in the first 2 rotations. They don't know where to look. So we have them act as a shadow, until they know the patterns.*

Lg6 also discussed an initiative at their company to reduce the alert fatigue, ironically with another model, which would consider how many times an engineer historically acted on an alert of a given type before determining whether to surface a new alert of that type.

4.4.2 ***Over time, ML pipelines may turn into "jungles" of rules and models.*** Sculley et al. [87] introduce the phrase "pipeline jungles" (i.e., different versions of data transformations and models glued together), which was later adopted by participants in our study. While prior work demonstrates their existence and maintenance challenges, we provide insight into why and how these pipelines become jungles in the first place. Our interviewees noted that reacting to an ML-related bug in production usually took a long time, motivating them to find strategies to quickly restore performance (Lg1, Sm2, Sm3, Sm4, Md4, Md5, Md6, Lg6). These strategies primarily involved adding non-ML rules and filters to the pipeline. When Sm3 observed, for an entity recognition task, that the model was misdetecting the Egyptian president due to the many ways of writing his name, they thought it would be better to patch the predictions for the individual case than to fix or retrain the model:

> *Suppose we deploy [a new model] in the place of the existing model. We'd have to go through all the training data and then relabel it and [expletive], that's so much work.*

One way engineers reacted to ML bugs was by adding filters for models. For the Egypt example, Sm3 added a simple string similarity rule to identify the president's name. Md4 and Md5 each discussed how their models were augmented with a final, rule-based layer to keep live predictions more stable. For example, Md4 mentioned working on an anomaly detection model and adding a heuristics layer on top to filter the set of anomalies that surface based on domain knowledge. Md5 discussed one of their language models for a customer support chatbot:

> *The model might not have enough confidence in the suggested reply, so we don't return [the recommendation]. Also, language models can say all sorts of things you don't necessarily want it to—another reason that we don't show some suggestions. For example, if somebody asks when the business is open, the model might try to quote a time when it thinks the business is open. [It might say] "9 am," but the model doesn't know that. So if we detect time, then we filter that [reply]. We have a lot of filters.*

Constructing such filters was an iterative process—Md5 mentioned constantly stress-testing the model in a sandbox, as well as observing suggested replies in early stages of deployment, to come up with filter ideas. Creating filters was a more effective strategy than trying to retrain the model to say the right thing; the goal was to keep some version of a model working in production with little downtime. As a result, filters would accumulate in the pipeline over time, effectively creating a pipeline jungle. Even when models were improved, Lg5 noted that it was too risky to remove the filters, since the filters were already in production, and a removal might lead to cascading or unforeseen failures.

Several engineers also maintained fallback models for reverting to: either older or simpler versions (Lg2, Lg3, Md6, Lg5, Lg6). Lg5 mentioned that it was important to always keep some model up and running, even if they "switched to a less economic model and had to just cut the losses." Similarly, when doing data science work, both Passi and Jackson [73] and Wang et al. [102] echo the importance of having some solution to meet clients' needs, even if it is not the best solution. Another simple solution engineers discussed was serving a separate model for each customer (Lg1, Lg3, Sm2, Sm4, Md3, Md4). We found that engineers preferred a per-customer model because it minimized downtime: if the service wasn't working for a particular customer, it could still work for other customers. Patel et al. [74] also noted that per-customer models could yield higher overall performance.

4.4.3 ***Bugs in production ML follow a heavy-tailed distribution.*** ML debugging is different from debugging during standard software engineering, where one can write test cases to cover the space of potential bugs [3, 71]. Lg3, Sm2, Sm3, Sm4, Lg4, Md4, Md5, Sm6, Lg5, and Lg6 mentioned having a *central queue of production ML bugs* that every engineer added tickets to and processed tickets from. Often this queue was larger than what engineers could process in a timely manner, so they assigned tags to tickets to prioritize what to debug.

Interviewees discussed ad-hoc approaches to debugging production ML issues, which could require them to spend a lot of time diagnosing any given bug (Lg3, Lg2, Sm3, Sm4, Lg5). One common issue was *data leakage*—i.e., assuming during training that there is access to data that does not exist at serving time—an error typically discovered after the model was deployed and several incorrect live predictions were made (Lg1, Md1, Md5, Lg5). Interviewees felt that anticipating all possible forms of data leakage during experimentation was tedious; thus, sometimes leakage was retroactively checked during code review in an evaluation stage (Lg1. There were other types of bugs that were discussed by multiple participants, such as accidentally flipping labels in classification models (Lg1, Sm1, Lg3, Md3) and forgetting to set random seeds in distributed training when initializing workers in parallel (Lg1, Lg4, Sm5). However, the vast majority of bugs described to us in the interviews were seemingly bespoke and not shared among participants. For example, Sm3 forgot to drop special characters (e.g., apostrophes) for their language models. Lg3 found that the imputation value for missing features was once corrupted. Lg5 mentioned that a feature of unstructured data type (e.g., JSON) had half of the keys' values missing for a "long time."

When asked how they detect these one-off bugs, interviewees mentioned that their bugs showed similar symptoms of failure. One symptom was a large discrepancy between offline validation accuracy and production accuracy immediately after deployment (Lg1, Lg3, Md4, Lg5). However, if there were no ground-truth labels available immediately after deployment (as discussed in Section 4.1.3), interviewees had to resort to other strategies. For example, some inspected the results of data quality checks (Section 4.4.1). Lg1 discussed their struggle to debug without "ground-truth:":

> *Um, yeah, it's really hard. Basically there's no surefire strategy. The closest that I've seen is for people to integrate a very high degree of observability into every part of their pipeline. It starts with having really good raw data, observability, and visualization tools. The ability to query. I've noticed, you know, so much of this [ad-hoc bug exploration] is just—if you make the friction [to debug] lower, people will do it more. So as an organization, you need to make the friction very low for investigating what the data actually looks like, [such as] looking at specific examples.*

To diagnose bugs, interviewees typically sliced and diced data for different groups of customers or data points (Md1, Lg3, Md3, Md4, Md6, Lg6). Slicing and dicing is known to be useful for identifying bias in models [31, 85], but we observed that our interviewees used this technique beyond debugging bias and fairness; they sliced and diced to determine common failure modes and

data points similar to these failures. Md4 discussed annotating bugs and only drilling down into their queue of bugs when they observed "systematic mistakes for a large number of customers."

Interviewees mentioned that after several iterations of chasing bespoke ML-related bugs in production, they had developed a sense of paranoia while evaluating models offline—possibly as a coping mechanism (Lg1, Md1, Lg3, Md5, Md6, Lg6). Lg1 said:

> *ML [bugs] don't get caught by tests or production systems and just silently cause errors. This is why [you] need to be paranoid when you're writing ML code, and then be paranoid when you're coding.*

Lg1 then recounted a bug that was "impossible to discover" after a deployment to production: the code for a change that added new data augmentation to the training procedure had two variables flipped, and this bug was miraculously caught during code review even though the training accuracy was high. Lg1 claimed that there was "no mechanism by which [they] would have found this besides someone just curiously reading the code." Since ML bugs don't cause systems to go down, sometimes the only way to find them is to be cautious when inspecting code, data, and models.

5 DISCUSSION

Our findings suggest that automated production ML pipelines are enabled by MLEs working through a continuous loop of i) data preparation, ii) experimentation, iii) evaluation & deployment, and iv) monitoring and response (Figure 1). Although engineers leverage different tools to help with technical aspects of their workflow, such as experiment tracking and data validation [8, 109], patterns began to emerge when we studied how MLE practices varied across company sizes and experience levels. We discuss these patterns as "the three Vs of MLOps" (Section 5.1), and follow our discussion with implications for both production ML tooling (Section 5.2), and opportunities for future work (Section 5.3).

5.1 Velocity, Visibility, Versioning: Three Vs of MLOps

Findings from our work and prior work suggest three broad themes of successful MLOps practices: Velocity, Visibility, and Versioning. These themes have synergies and tensions across each stage of MLEs' workflow, as we discuss next.

5.1.1 ***Velocity.*** Since ML is so experimental in nature, it's important to be able to prototype and iterate on ideas quickly (e.g., go from a new idea to a trained model in a day). Interviewees attributed their productivity to development environments that prioritized high experimentation velocity and debugging environments that allowed them to test hypotheses quickly. Prior work has extensively documented the Agile tendencies of MLEs, describing how they iterate quickly (i.e. with *velocity*) to explore a large ML or data science search space [3, 30, 50, 74, 110]. Amershi et al. [3] describe how experimentation can be sped up when labels are annotated faster (i.e., rapid data preparation). Garcia et al. [20] explore tooling to help MLEs correct logging oversights from too much velocity in experimentation, and Paleyes et al. [71] mention the need to diagnose production bugs quickly to prevent future similar issues from occurring. First, our study re-enforces the view the MLEs are agile workers who value fast results. P1 said that people who achieve the best outcomes from experimentation are people with "scary high experimentation velocity." Similarly, the multi-stage deployment strategy can be viewed as an optimistic or high-velocity solution to deployment: deploy first, and validate gradually across stages. Moreover, our study provides deeper insight into how practitioners rapidly debug deployments—we identify and describe practices such as on-call rotations, human-interpretable filters on model behavior, data quality alerts, and model rollbacks.

At the same time, high velocity can lead to trouble if left unchecked. Sambasivan et al. [85] observed that, for high-stakes customers, practitioners iterated too quickly, causing ML systems to fail—practitioners "moved fast, hacked model performance (through hyperparameters rather than data quality), and did not appear to be equipped to recognise upstream and downstream people issues." Our study exposed strategies that practitioners used to prevent themselves from iterating too quickly: for example, in Section 4.3.1, we described how some applications (e.g., autonomous vehicles) require separate teams to manage evaluation, making sure that bad models don't get promoted from development to production. Moreover, when measuring ML metrics outside of accuracy, e.g., fairness [31] or product metrics (Section 4.3.2), it is challenging to make sure all metrics improve for each change to the ML pipeline [71]. Understanding which metrics to prioritize requires domain and business expertise [40], which could hinder velocity.

5.1.2 ***Visibility.*** In organizations, since many stakeholders and teams are impacted by ML-powered applications and services, it is important for MLEs to have an end-to-end view of ML pipelines. P1 explicitly mentioned integrating "very high degree of observability into every part of [the] pipeline" (Section 4.4.3). Prior work describes the importance of visibility: for example, telemetry data from ML pipelines (e.g., logs and traces) can help engineers know if the pipeline is broken [40], model explainability methods can establish customers' trust in ML predictions [42, 71, 102], and dashboards on ML pipeline health can help align nontechnical stakeholders with engineers [37, 46]. In our view, the popularity of Jupyter notebooks among MLEs, including among the participants in our study, can be explained by Jupyter's gains in velocity and visibility for ML experimentation, as it effectively combines REPL (Read-Eval-Print-Loop)-style interaction and visualization capabilities despite its *versioning* shortcomings. Our findings corroborate these prior findings and provide further insight on how visibility is achieved in practice. For example, engineers proactively monitor feedback delays (Section 4.1.3). They also document live failures frequently to keep validation datasets current (Section 4.3.1), and they engage in on-call rotations to investigate data quality alerts (Section 4.4).

Visibility also helps with velocity. If engineers can quickly identify the source of a bug, they can fix it faster. Or, if other stakeholders, such as product managers or business analysts, can understand how an experiment or multi-staged deployment is progressing, they can better use their domain knowledge to assess models according to product metrics (see Section 4.3.2), and intervene sooner if there's evidence of a problem. One of the pain points we observed was that end-to-end experimentation—from the conception of an idea to improve ML performance to validation of the idea—took too long. The uncertainty of project success stems from the unpredictable, real-world nature of experiments.

5.1.3 ***Versioning.*** Amershi et al. [3] mention that "fast-paced model iteration" requires careful versioning of data and code. Other work identifies a need to also manage model versions [101, 109]. Our work suggests that mananging *all* artifacts—data, code, models, data quality metrics, filters, rules—in tandem is extremely challenging but vital to the success of ML deployments. Prior work explains how these artifacts can be queried during debugging [7, 11, 87], and our findings additionally show that versioning is particularly useful when *teams* of people work on ML pipelines. For instance, during monitoring, on-call engineers may receive a flood of false-positive alerts; looking at old alerts might help them understand whether a specific type of alert actually requires action. In another example, during experimentation, ML engineers often work on models and pipelines they didn't initially create. Versioning increases visibility: engineers can inspect old versions of experiments to understand ideas that may or may not have worked.

Not only does versioning aid visibility, but it also enables workflows to maintain high velocity. In Section 4.4.2, we explained how pipeline jungles are created by quickly responding to ML bugs

by constructing various filters and rules. If engineers had to fix the training dataset or model for every bug, they would not be able to iterate quickly. Maintaining different versions for different types of inputs (e.g., rules to auto-reject incomplete data or different models for different users) also enables high velocity. However, there is also a tension between velocity and versioning: in Section 4.2.3, we discussed how parallelizing experiment ideas produces many versions, and ML engineers could not cognitively keep track of them. In other words, having high velocity can mean drowning in a sea of versions.

5.2 Opportunities for ML Tooling

Our main takeaway is that production ML tooling needs to aid *humans* in their workflows, not just automate technical practices (e.g., generating a feature or training a model). Tools should help improve at least one of the three Vs, and there are opportunities for tools in each stage of the workflow. We discuss each in turn.

5.2.1 Data Preparation. As mentioned in Section 4.1, separate teams of data engineers typically manage pipelines to ingest, clean, and preprocess data on a schedule. While existing tools automate scheduling these activities, there are unadressed ML needs around retraining and labeling. Prior work and our interviews indicate that ML engineers retrain models on some arbitrary cadence [44, 71], without understanding the effect of the cadence on the quality of predictions. Models might be stale if they are retrained only monthly, or they might retrain using invalid or corrupt data if they are retrained faster than the data is validated and cleaned (e.g., hourly). Moreover, the optimal retraining cadence depends on the data, ML task, and amount of organizational resources, such as compute, training time, and number of engineers on the team. New tools can help with these challenges and determine the best retraining cadence for ML pipelines. With respect to labeling, existing tools help with either labeling at scale [79] or labeling with high quality [45], but it is hard to achieve both. As a result, organizations have custom infrastructure and teams to resolve label mismatches, apply domain knowledge, and reject incorrect labels. Labeling tools can leverage ensembling and add postprocessing filters to reject and resolve incorrect and inconsistent labels. Moreover, they should track feedback delays and surface this information to users.

5.2.2 Experimentation. As discussed in Section 4.2.3, experiments are typically done in development environments and then promoted to production clusters during deployment. The mismatch between the two (or more!) environments can cause bugs, creating an opportunity for new tools. The development cluster should maximize iteration speed (velocity), while the production cluster should minimize end-user prediction latency [14]. Hardware and software can be different in each cluster, e.g., GPUs for training vs. CPUs for inference, and Python vs. C++, which makes this problem challenging. New tools are prioritizing reproducibility—turning Jupyter notebooks into production scripts [90], for instance—but should also standardize how engineers interact with experimentation workflows. For example, while experiment tracking tools can literally keep track of thousands of experiments, how can engineers sort through all these versions and actually understand what the best experiments are doing? Our findings and prior work show that the experimental nature of ML and data science leads to undocumented tribal knowledge within organizations [37, 41]. Documentation solutions for deployed models and datasets have been proposed [23, 60], but we see an opportunity for tools to help document *experiments*—particularly, failed ones. Forcing engineers to write down institutional knowledge about what ideas work or don't work slows them down, and automated documentation assistance would be quite useful.

5.2.3 Evaluation and Deployment. Prior work has identified several opportunities in the evaluation and deployment space. For example, there is a need to map ML metric gains to product or business

gains [40, 44, 57]. Additionally, tools could help define and calculate subpopulation-specific performance metrics [31]. From our study, we have observed a need for tooling around the multi-staged deployment process. With multiple stages, the turnaround time from experiment idea to having a full production deployment (i.e., deployed to all users) can take several months. Invalidating ideas in earlier stages of deployment can increase overall, end-to-end velocity. Our interviewees discussed how some feature ideas no longer make sense after a few months, given the nature of how user behaviors change, which would cause an initially good idea to never fully and finally deploy to production. Additionally, an organization's key product metrics—e.g., revenue or number of clicks—might change in the middle of a multi-stage deployment, killing the deployment. This negatively impacts the engineers responsible for the deployment. We see this as an opportunity for new tools to streamline ML deployments in this multi-stage pattern, to minimize wasted work and help practitioners predict the end-to-end gains for their ideas.

5.2.4 Monitoring and Response. Recent work in ML observability identifies a need for tools to give end-to-end visibility on ML pipeline behavior and debug ML issues faster [7, 91]. Basic data quality statistics, such as missing data and type or schema checks, fail to capture anomalies in the values of data [8, 62, 76]. Our interviewees complained that existing tools that attempt to flag anomalies in the values of data points produce too many false positives (Section 4.4.1). An excessive number of false-positive alerts, i.e., data points flagged as invalid even if they are valid, leads to two pain points: (1) unnecessarily maintaining many model versions or simple heuristics for invalid data points, which can be hard to keep track of, and (2) a lower overall accuracy or ML metric, as baseline models might not serve high-quality predictions for these invalid points. Moreover, due to feedback delays, it may not be possible to track ML performance (e.g., accuracy) in real time. What metrics can be reliably monitored in real time, and what criteria should trigger alerts to maximize precision and recall when identifying model performance drops? How can these metrics and alerting criteria automatically tune themselves over time, as the underlying data changes? We envision this to be an opportunity for new data management tools.

Moreover, as discussed in Section 4.4.2, when engineers quickly respond to production bugs, they create pipeline jungles. Such jungles typically consist of several versions of models, rules, and filters. Most of the ML pipelines that our interviewees discussed were pipeline jungles. This combination of modern model-driven ML and old-fashioned rule-based AI indicates a need for managing filters (and versions of filters) in addition to managing learned models. The engineers we interviewed managed these artifacts themselves.

5.3 Limitations and Future Work

Since we wanted to find common themes in production ML workflows across different applications and organizations, our interview study's scope was quite broad: we set out on a quest to discover shared patterns, rather than to predict or explain. We asked practitioners open-ended questions about their MLOps workflows and challenges, but did not probe them about questions of fairness, risk, and data governance: these questions could be studied in future interviews. Moreover, we did not focus on the *differences* between practitioners' workflows based on their company sizes, educational backgrounds, or industries. While there are interview studies for specific applications of ML [6, 19, 77], we see further opportunities to study the effect of organizational focus and maturity on the production ML workflow. There are also questions for which interview studies are a poor fit. Given our findings on the importance of collaborative and social dimensions of MLOps, we would like to explore these ideas further through participant action research or contextual inquiry.

Moreover, our paper focuses on a *human-centered* workflow surrounding production ML pipelines. Focusing on the *automated* workflows in ML pipelines—for example, continuous integration and

continuous deployment (CI/CD)—could prove a fruitful research direction. Finally, we only interviewed ML engineers, not other stakeholders, such as software engineers or product managers. Kreuzberger et al. [44] present a diagram of technical components of the ML pipeline (e.g., feature engineering, model training) and interactions between ML engineers and other stakeholders. Another interview study could observe these interactions and provide further insight into practitioners' workflows.

6 CONCLUSION

In this paper, we presented results from a semi-structured interview study of 18 ML engineers spanning different organizations and applications to understand their workflow, best practices, and challenges. Engineers reported several strategies to sustain and improve the performance of production ML pipelines, and we identified four stages of their MLOps workflow: i) data preparation, ii) experimentation, iii) evaluation and deployment, and iv) monitoring and response. Throughout these stages, we found that successful MLOps practices center around having good velocity, visibility, and versioning. Finally, we discussed opportunities for tool development and research.

ACKNOWLEDGMENTS

This is the acknowledgement

REFERENCES

[1] Leonel Aguilar, David Dao, Shaoduo Gan, Nezihe Merve Gurel, Nora Hollenstein, Jiawei Jiang, Bojan Karlas, Thomas Lemmin, Tian Li, Yang Li, Susie Rao, Johannes Rausch, Cedric Renggli, Luka Rimanic, Maurice Weber, Shuai Zhang, Zhikuan Zhao, Kevin Schawinski, Wentao Wu, and Ce Zhang. 2021. Ease.ML: A Lifecycle Management System for MLDev and MLOps. In *Conference on Innovative Data Systems Research (CIDR 2021)*. **https://www.microsoft.com/en-us/research/publication/ease-ml-a-lifecycle-management-system-for-mldev-and-mlops/**

[2] Sridhar Alla and Suman Kalyan Adari. 2021. What is mlops? In *Beginning MLOps with MLFlow*. Springer, 79–124.

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

[4] Anonymous. 2021. ML Reproducibility Systems: Status and Research Agenda. **https://openreview.net/forum?id=v-6XBItNld2**

[5] Amitabha Banerjee, Chien-Chia Chen, Chien-Chun Hung, Xiaobo Huang, Yifan Wang, and Razvan Chevesaran. 2020. Challenges and Experiences with {MLOps} for Performance Diagnostics in {Hybrid-Cloud} Enterprise Software Deployments. In *2020 USENIX Conference on Operational Machine Learning (OpML 20)*.

[6] Catherine Billington, Gonzalo Rivero, Andrew Jannett, and Jiating Chen. 2022. A Machine Learning Model Helps Process Interviewer Comments in Computer-assisted Personal Interview Instruments: A Case Study. *Field Methods* (2022), 1525822X221107053.

[7] Mike Brachmann, Carlos Bautista, Sonia Castelo, Su Feng, Juliana Freire, Boris Glavic, Oliver Kennedy, Heiko Müeller, Rémi Rampin, William Spoth, and Ying Yang. 2019. Data Debugging and Exploration with Vizier. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1877–1880. **https://doi.org/10.1145/3299869.3320246**

[8] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. 2019. Data Validation for Machine Learning. In *Proceedings of SysML*. **https://mlsys.org/Conferences/2019/doc/2019/167.pdf**

[9] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. 2006. VisTrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. 745–747.

[10] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. 1999. The CRISP-DM user guide. In *4th CRISP-DM SIG Workshop in Brussels in March*, Vol. 1999. sn.

[11] Amit Chavan, Silu Huang, Amol Deshpande, Aaron Elmore, Samuel Madden, and Aditya Parameswaran. 2015. Towards a unified query language for provenance and versioning. In *7th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 15)*.

[12] Ji Young Cho and Eun-Hee Lee. 2014. Reducing confusion about grounded theory and qualitative content analysis: Similarities and differences. *Qualitative report* 19, 32 (2014).

[13] David Cohen, Mikael Lindvall, and Patricia Costa. 2004. An introduction to agile methods. *Adv. Comput.* 62, 03 (2004), 1–66.

[14] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A {Low-Latency} Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 613–627.

[15] John W Creswell and Cheryl N Poth. 2016. *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications.

[16] Susan B Davidson and Juliana Freire. 2008. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 1345–1350.

[17] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. DevOps. *Ieee Software* 33, 3 (2016), 94–100.

[18] Mihail Eric. [n. d.]. MLOps is a mess but that's to be expected. **https://www.mihaileric.com/posts/mlops-is-a-mess/**

[19] Asbjørn Følstad, Cecilie Bertinussen Nordheim, and Cato Alexander Bjørkli. 2018. What makes users trust a chatbot for customer service? An exploratory interview study. In *International conference on internet science*. Springer, 194–208.

[20] Rolando Garcia, Eric Liu, Vikram Sreekanti, Bobby Yan, Anusha Dandamudi, Joseph E. Gonzalez, Joseph M Hellerstein, and Koushik Sen. 2021. Hindsight Logging for Model Training. In *VLDB*.

[21] Rolando Garcia, Vikram Sreekanti, Neeraja Yadwadkar, Daniel Crankshaw, Joseph E Gonzalez, and Joseph M Hellerstein. 2018. Context: The missing piece in the machine learning lifecycle. In *CMI*.

[22] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg, and Saransh Ahlawat. 2021. On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps. In *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. 25–28. **https://doi.org/10.1109/AIKE52691.2021.00010**

[23] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92.

[24] Samadrita Ghosh. 2021. Mlops challenges and how to face them. **https://neptune.ai/blog/mlops-challenges-and-how-to-face-them**

[25] Tuomas Granlund, Aleksi Kopponen, Vlad Stirbu, Lalli Myllyaho, and Tommi Mikkonen. 2021. MLOps challenges in multi-organization setup: Experiences from two real-world cases. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 82–88.

[26] Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How many interviews are enough? An experiment with data saturation and variability. *Field methods* 18, 1 (2006), 59–82.

[27] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 65–74. **https://doi.org/10.1145/2047196.2047205**

[28] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, et al. 2017. Ground: A Data Context Service.. In *CIDR*.

[29] Charles Hill, Rachel K. E. Bellamy, Thomas Erickson, and Margaret M. Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2016), 162–170.

[30] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and visualizing data iteration in machine learning. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–13.

[31] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé, Miro Dudik, and Hanna Wallach. 2019. Improving Fairness in Machine Learning Systems: What Do Industry Practitioners Need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. ACM Press, Glasgow, Scotland Uk, 1–16. **https://doi.org/10.1145/3290605.3300830**

[32] Youyang Hou and Dakuo Wang. 2017. Hacking with NPOs: Collaborative Analytics and Broker Roles in Civic Data Hackathons. *Proc. ACM Hum.-Comput. Interact.* 1, CSCW, Article 53 (dec 2017), 16 pages. **https://doi.org/10.1145/3134688**

[33] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R Pocock, Peter Li, and Tom Oinn. 2006. Taverna: a tool for building and running workflows of services. *Nucleic acids research* 34, suppl_2 (2006), W729–W732.

[34] Chip Huyen. 2020. Machine learning tools landscape V2 (+84 new tools). **https://huyenchip.com/2020/12/30/mlops-v2.html**

[35] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. 2021. Towards mlops: A framework and maturity model. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 1–8.

[36] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3363–3372.

[37] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2917–2926. **https://doi.org/10.1109/TVCG.2012.219**

[38] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. [n. d.]. Model assertions for debugging machine learning.

[39] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.

[40] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The Emerging Role of Data Scientists on Software Development Teams. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 96–107. **https://doi.org/10.1145/2884781.2884783**

[41] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1024–1038.

[42] Janis Klaise, Arnaud Van Looveren, Clive Cox, Giovanni Vacanti, and Alexandru Coca. 2020. Monitoring and explainability of models in production. *ArXiv* abs/2007.06299 (2020).

[43] Johannes Köster and Sven Rahmann. 2012. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* 28, 19 (2012), 2520–2522.

[44] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. 2022. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. **https://doi.org/10.48550/ARXIV.2205.02302**

[45] Sanjay Krishnan and Eugene Wu. 2017. PALM: Machine Learning Explanations For Iterative Debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics - HILDA'17*. ACM Press, Chicago, IL, USA, 1–6. **https://doi.org/10.1145/3077257.3077271**

[46] Sean Kross and Philip Guo. 2021. Orienting, Framing, Bridging, Magic, and Counseling: How Data Scientists Navigate the Outer Loop of Client Collaborations in Industry and Academia. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 311 (oct 2021), 28 pages. **https://doi.org/10.1145/3476052**

[47] Sean Kross and Philip J Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–14.

[48] Indika Kumara, Rowan Arts, Dario Di Nucci, Willem Jan Van Den Heuvel, and Damian Andrew Tamburri. 2022. Requirements and Reference Architecture for MLOps: Insights from Industry. (2022).

[49] Sampo Kuutti, R. Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. 2021. A Survey of Deep Learning Applications to Autonomous Vehicle Control. *IEEE Transactions on Intelligent Transportation Systems* 22 (2021), 712–733.

[50] Angela Lee, Doris Xin, Doris Lee, and Aditya Parameswaran. 2020. Demystifying a Dark Art: Understanding Real-World Machine Learning Model Development. **https://doi.org/10.48550/ARXIV.2005.01520**

[51] Cheng Han Lee. 2020. 3 data careers decoded and what it means for you. **https://www.udacity.com/blog/2014/12/data-analyst-vs-data-scientist-vs-data-engineer.html**

[52] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–35.

[53] Anderson Lima, Luciano Monteiro, and Ana Paula Furtado. 2022. MLOps: Practices, Maturity Models, Roles, Tools, and Challenges-A Systematic Literature Review. *ICEIS (1)* (2022), 308–320.

[54] Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. 2021. The CLEAR Benchmark: Continual LEArning on Real-World Imagery. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. **https://openreview.net/forum?id=43mYF598ZDB**

[55] Mike Loukides. 2012. *What is DevOps?* " O'Reilly Media, Inc.".

[56] Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo, and Casper Lassenius. 2019. DevOps in practice: A multiple case study of five companies. *Information and Software Technology* 114 (2019), 217–230.

[57] Michael A. Madaio, Luke Stark, Jennifer Wortman Vaughan, and Hanna Wallach. 2020. Co-Designing Checklists to Understand Organizational Challenges and Opportunities around Fairness in AI. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. **https://doi.org/10.1145/3313831.3376445**

[58] Sasu Mäkinen, Henrik Skogström, Eero Laaksonen, and Tommi Mikkonen. 2021. Who needs MLOps: What data scientists seek to accomplish and how can MLOps help?. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software*

Engineering for AI (WAIN). IEEE, 109–112.
[59] Beatriz MA Matsui and Denise H Goya. 2022. MLOps: five steps to guide its effective implementation. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. 33–34.
[60] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.
[61] MLReef. 2021. Global mlops and ML Tools Landscape: Mlreef. **https://about.mlreef.com/blog/global-mlops-and-ml-tools-landscape/**
[62] Akshay Naresh Modi et al. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *KDD 2017*.
[63] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognition* 45, 1 (2012), 521–530. **https://doi.org/10.1016/j.patcog.2011.06.019**
[64] Dennis Muiruri, Lucy Ellen Lwakatare, Jukka K Nurminen, and Tommi Mikkonen. 2022. Practices and Infrastructures for ML Systems–An Interview Study in Finnish Organizations. (2022).
[65] Michael Muller. 2014. Curiosity, creativity, and surprise as analytic tools: Grounded theory method. In *Ways of Knowing in HCI*. Springer, 25–48.
[66] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How data science workers work with data: Discovery, capture, curation, design, creation. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–15.
[67] Mohammad Hossein Namaki, Avrilia Floratou, Fotis Psallidas, Subru Krishnan, Ashvin Agrawal, Yinghui Wu, Yiwen Zhu, and Markus Weimer. 2020. Vamsa: Automated provenance tracking in data science scripts. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1542–1551.
[68] Andrew Ng, Eddy Shyu, Aarti Bagul, and Geoff Ladwig. [n. d.]. Evaluating a model - advice for applying machine learning. **https://www.coursera.org/lecture/advanced-learning-algorithms/evaluating-a-model-26yGi**
[69] Yaniv Ovadia, Emily Fertig, J. Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift. In *NeurIPS*.
[70] Cosmin Paduraru, Daniel J. Mankowitz, Gabriel Dulac-Arnold, Jerry Li, Nir Levine, Sven Gowal, and Todd Hester. 2021. Challenges of Real-World Reinforcement Learning:Definitions, Benchmarks & Analysis. *Machine Learning Journal* (2021).
[71] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Comput. Surv.* (apr 2022). **https://doi.org/10.1145/3533378** Just Accepted.
[72] Samir Passi and Steven J. Jackson. 2017. Data Vision: Learning to See Through Algorithmic Abstraction. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (2017).
[73] Samir Passi and Steven J Jackson. 2018. Trust in data science: Collaboration, translation, and accountability in corporate data science projects. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–28.
[74] Kayur Patel, James Fogarty, James A. Landay, and Beverly L. Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *International Conference on Human Factors in Computing Systems*.
[75] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1723–1726.
[76] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data Lifecycle Challenges in Production Machine Learning: A Survey. *SIGMOD Record* 47, 2 (2018), 12.
[77] Luisa Pumplun, Mariska Fecho, Nihal Wahl, Felix Peters, Peter Buxmann, et al. 2021. Adoption of machine learning systems for medical diagnostics in clinics: qualitative interview study. *Journal of Medical Internet Research* 23, 10 (2021), e29301.
[78] Stephan Rabanser, Stephan Günnemann, and Zachary Chase Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *NeurIPS*.
[79] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 11. NIH Public Access, 269.
[80] Gilberto Recupito, Fabiano Pecorelli, Gemma Catolino, Sergio Moreschini, Dario Di Nucci, Fabio Palomba, and Damian A Tamburri. 2022. A multivocal literature review of mlops tools and features. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 84–91.

[81] Cedric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlaš, Wentao Wu, and Ce Zhang. 2021. A data quality-driven view of mlops. *arXiv preprint arXiv:2102.07750* (2021).

[82] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2019. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering* (2019), 1–1. **https://doi.org/10.1109/TKDE.2019.2946162** Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[83] Philipp Ruf, Manav Madan, Christoph Reich, and Djaffar Ould-Abdeslam. 2021. Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences* 11, 19 (2021), 8861.

[84] Lukas Rupprecht, James C Davis, Constantine Arnold, Yaniv Gur, and Deepavali Bhagwat. 2020. Improving reproducibility of data science pipelines through transparent provenance capture. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3354–3368.

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

[86] Sebastian Schelter et al. 2018. Automating Large-Scale Data Quality Verification. In *PVLDB'18*.

[87] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *NIPS*.

[88] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. 2020. Adoption and Effects of Software Engineering Best Practices in Machine Learning. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Bari, Italy) *(ESEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 3, 12 pages. **https://doi.org/10.1145/3382494.3410681**

[89] Shreya Shankar, Bernease Herman, and Aditya G. Parameswaran. 2022. Rethinking Streaming Machine Learning Evaluation. *ArXiv* abs/2205.11473 (2022).

[90] Shreya Shankar, Stephen Macke, Sarah Chasins, Andrew Head, and Aditya Parameswaran. 2023. Bolt-on, Compact, and Rapid Program Slicing for Notebooks. *Proc. VLDB Endow.* (sep 2023).

[91] Shreya Shankar and Aditya G. Parameswaran. 2022. Towards Observability for Production Machine Learning Pipelines. *ArXiv* abs/2108.13557 (2022).

[92] James P Spradley. 2016. *The ethnographic interview*. Waveland Press.

[93] Megha Srivastava, Besmira Nushi, Ece Kamar, S. Shah, and Eric Horvitz. 2020. An Empirical Analysis of Backward Compatibility in Machine Learning Systems. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2020).

[94] Steve Nunez. 2022. Why AI investments fail to deliver. **https://www.infoworld.com/article/3639028/why-ai-investments-fail-to-deliver.html** [Online; accessed 15-September-2022].

[95] Anselm Strauss and Juliet Corbin. 1994. Grounded theory methodology: An overview. (1994).

[96] Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Müller. 2021. Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology. *Machine learning and knowledge extraction* 3, 2 (2021), 392–413.

[97] Masashi Sugiyama et al. 2007. Covariate Shift Adaptation by Importance Weighted Cross Validation. In *JMLR*.

[98] Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis, and George A. Papakostas. 2022. MLOps - Definitions, Tools and Challenges. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 0453–0460. **https://doi.org/10.1109/CCWC54503.2022.9720902**

[99] Damian A Tamburri. 2020. Sustainable mlops: Trends and challenges. In *2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC)*. IEEE, 17–23.

[100] Matteo Testi, Matteo Ballabio, Emanuele Frontoni, Giulio Iannello, Sara Moccia, Paolo Soda, and Gennaro Vessio. 2022. MLOps: A taxonomy and a methodology. *IEEE Access* 10 (2022), 63606–63618.

[101] Manasi Vartak. 2016. ModelDB: a system for machine learning model management. In *HILDA '16*.

[102] Dakuo Wang, Justin D. Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI Collaboration in Data Science: Exploring Data Scientists' Perceptions of Automated AI. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 211 (nov 2019), 24 pages. **https://doi.org/10.1145/3359313**

[103] Joyce Weiner. 2020. Why AI/data science projects fail: how to avoid project pitfalls. *Synthesis Lectures on Computation and Analytics* 1, 1 (2020), i–77.

[104] Wikipedia contributors. 2022. MLOps — Wikipedia, The Free Encyclopedia. **https://en.wikipedia.org/w/index.php?title=MLOps&oldid=1109828739** [Online; accessed 15-September-2022].

[105] Olivia Wiles, Sven Gowal, Florian Stimberg, Sylvestre-Alvise Rebuffi, Ira Ktena, Krishnamurthy Dvijotham, and Ali Taylan Cemgil. 2021. A Fine-Grained Analysis on Distribution Shift. *ArXiv* abs/2110.11328 (2021).

[106] Kanit Wongsuphasawat, Yang Liu, and Jeffrey Heer. 2019. Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. *ArXiv* abs/1911.00568 (2019).

[107] Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. In *Proceedings of the 2021 International Conference on Management of Data*. 2639–2652.

[108] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. 2021. Whither AutoML? Understanding the Role of Automation in Machine Learning Workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 83, 16 pages. https://doi.org/10.1145/3411764.3445306

[109] M. Zaharia et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41 (2018), 39–45.

[110] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.

[111] Yu Zhang, Yun Wang, Haidong Zhang, Bin Zhu, Siming Chen, and Dongmei Zhang. 2022. OneLabeler: A Flexible System for Building Data Labeling Tools. In *CHI Conference on Human Factors in Computing Systems*. 1–22.

A SEMI-STRUCTURED INTERVIEW QUESTIONS

In the beginning of each interview, we explained the purpose of the interview—to better understand processes within the organization for validating changes made to production ML models, ideally through stories of ML deployments. We then kickstarted the information-gathering process with a question to build rapport with the interviewee, such as *tell us about a memorable previous ML model deployment*. This question helped us isolate an ML pipeline or product to discuss. We then asked a series of open-ended questions:

(1) **Nature of ML task**
- What is the ML task you are trying to solve?
- Is it a classification or regression task?
- Are the class representations balanced?

(2) **Modeling and experimentation ideas**
- How do you come up with experiment ideas?
- What models do you use?
- How do you know if an experiment idea is good?
- What fraction of your experiment ideas are good?

(3) **Transition from development to production**
- What processes do you follow for promoting a model from the development phase to production?
- How many pull requests do you make or review?
- What do you look for in code reviews?
- What automated tests run at this time?

(4) **Validation datasets**
- How did you come up with the dataset to evaluate the model on?
- Do the validation datasets ever change?
- Does every engineer working on this ML task use the same validation datasets?

(5) **Monitoring**
- Do you track the performance of your model?
- If so, when and how do you refresh the metrics?
- What information do you log?
- Do you record provenance?
- How do you learn of an ML-related bug?

(6) **Response**
- What historical records (e.g., training code, training set) do you inspect in the debugging process?
- What organizational processes do you have for responding to ML-related bugs?
- Do you make tickets (e.g., Jira) for these bugs?
- How do you react to these bugs?
- When do you decide to retrain the model?

B TOOLS REFERENCED IN INTERVIEWS

Table 3 lists several of the tools that were commonly referenced by the interviewees.

Received 15 January 2023

	Data Collection	Experimentation	Evaluation and Deployment	Monitoring and Response
Metadata	Data catalogs, Amundsen, AWS Glue, Hive metastores	Weights & Biases, MLFlow, train/test set parameter configs, A/B test tracking tools		Dashboards, SQL, metric functions and window sizes
Unit	Data cleaning tools	Tensorflow, MLlib, PyTorch, Scikit-learn, XGBoost	OctoML, TVM, joblib, pickle	Scikit-learn metric functions, Great Expectations, Deequ
	Python, Pandas, Spark, SQL, C++, ONNX			
Pipeline	In-house or outsourced annotators	AutoML	Github Actions, Travis CI, Prediction serving tools, Kafka, Flink	Prometheus, AWS CloudWatch
	Airflow, Kubeflow, Argo, Tensorflow Extended (TFX), Vertex AI, DBT			
Infrastructure	Annotation schema, cleaning criteria configs	Jupyter notebook setups, GPUs	Edge devices, CPUs	Logging and observability services (e.g., DataDog)
	Cloud (e.g., AWS, GCP), compute clusters, storage (e.g., AWS S3, Snowflake), Docker, Kubernetes			

Table 3. Common tools referenced in interview transcripts, segmented by stage in the MLOps workflow and layer in the stack. The metadata layer is concerned with artifacts for component runs, like results of a training script. The unit layer represents individual pieces or components of a pipeline, such as feature engineering or model training. The pipeline layer connects components through orchestration frameworks, and the lowest layer is the infrastructure (e.g., compute).